2016-06

Using posture estimation to enhance personal inertial tracking

Foushee, Adam E.

Monterey, California: Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

USING POSTURE ESTIMATION TO ENHANCE PERSONAL INERTIAL TRACKING

by

Adam E. Foushee

June 2016

Thesis Advisor:	Xiaoping Yun
Co-Advisor:	Zachary Staples
Second Reader:	James Calusdian

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704–0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2016	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE USING POSTURE ESTIMATION TO ENHANCE PERSONAL INERTIAL TRACKING	**5. FUNDING NUMBERS**
6. AUTHOR(S) Adam E. Foushee	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

In close quarters combat, the lack of situational awareness can cause confusion, limit tempo of operations, and lead to fratricide. One approach to enhance the small-unit leader's situational awareness is to develop a network capable of mapping friendly positions. Current techniques for tracking the individual rifleman rely heavily on GPS, which does not work well indoors where satellite coverage is limited or even non-existent. One solution is to use inertial navigation systems to augment tracking during periods without GPS coverage. The goal of this research is to improve the current personal inertial navigation system by reducing or eliminating drift errors that are prevalent in this technology. The posture-tracking algorithm uses the YEI 3-space Data-Logging sensors to compute the posture of the individual rifleman. By tracking posture, stationary periods can be detected, and drift errors in the inertial navigation system are reduced. In the testing phase, the posture estimation algorithm was integrated with the personal navigation system, which is currently under development by concurrent research at the Naval Postgraduate School. Increased accuracy for inertial navigation systems that include posture tracking are demonstrated by the results of this thesis.

14. SUBJECT TERMS Personal navigation system, rotation matrix, geometry of fire, situational awareness, close quarters combat, orientation, coordinate system, reference frame, quaternion, Euler angles, YEI, posture tracking, human pose.	**15. NUMBER OF PAGES** 81
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540–01-280-5500 Standard Form 298 (Rev. 2–89) Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

USING POSTURE ESTIMATION TO ENHANCE PERSONAL INERTIAL TRACKING

Adam E. Foushee
Major, United States Marine Corps
B.S., University of North Carolina at Wilmington, 2005

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING
from the

NAVAL POSTGRADUATE SCHOOL
June 2016

Approved by: Xiaoping Yun
Thesis Advisor

Zachary Staples
Co-Advisor

James Calusdian
Second Reader

R. Clark Robertson
Chair, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In close quarters combat, the lack of situational awareness can cause confusion, limit tempo of operations, and lead to fratricide. One approach to enhance the small-unit leader's situational awareness is to develop a network capable of mapping friendly positions. Current techniques for tracking the individual rifleman rely heavily on GPS, which does not work well indoors where satellite coverage is limited or even non-existent. One solution is to use inertial navigation systems to augment tracking during periods without GPS coverage. The goal of this research is to improve the current personal inertial navigation system by reducing or eliminating drift errors that are prevalent in this technology. The posture-tracking algorithm uses the YEI 3-space Data-Logging sensors to compute the posture of the individual rifleman. By tracking posture, stationary periods can be detected, and drift errors in the inertial navigation system are reduced. In the testing phase, the posture estimation algorithm was integrated with the personal navigation system, which is currently under development by concurrent research at the Naval Postgraduate School. Increased accuracy for inertial navigation systems that include posture tracking are demonstrated by the results of this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

BFT	Blue Force Tracker
CQC	close quarters combat
FQA	factored quaternion algorithm
GPS	Global Positioning System
IMU	inertial measurement unit
MARG	magnetic, angular rate and gravity
NED	north east down
PNS	personal navigation system
VBSS	visit, board, search and seizure

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

This thesis provided an opportunity for me to use my knowledge gained at the Naval Postgraduate School and apply it to real-world problems related to my career in the military. The advancements made here could eventually save lives. This work could not have been possible without help from my advisor, Dr. Xiaoping Yun, my co-advisor, Commander Zachary Staples, or my second reader, Dr. James Calusdian, who spent numerous hours in the lab helping me de-bug and fix my programs. I would also like to thank my wife and children who offered encouragement and allowed me the time away from home to work on my research.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. CLOSE QUARTERS COMBAT AND OTHER GPS DENIED ENVIRONMENTS

Combat units have many complex regimes in which they fight. From open deserts to urban environments, units fight in many challenging places. Close quarters combat (CQC) and visit, board, search and seizure (VBSS) operations are among the most challenging tasks that a unit can perform. At all times, combat units must maintain a high level of situational awareness in order to keep the tempo of operations high. Knowing where each element of the unit is located, and even where each individual rifleman is located, can be a daunting task for a leader. For large items such as vehicles or aircraft, a Global Positioning System (GPS) is used to track the locations of friendly units and display these positions on a Blue Force Tracker (BFT). With the current technology, it is still impractical to use this technique for tracking human movement. The problem becomes more complex in the CQC or VBSS environment because GPS coverage is unreliable and can be affected by the enemy.

For infantry units, situational awareness about the location of friendly forces is important for preventing fratricide. For example, when an individual rifleman is inside of a building clearing a room, how does he know what or who is behind the wall? The small-unit leader needs a way to increase the tempo of his unit while also maintaining good geometries of fire. Currently, units train in the CQC environment to practice de-confliction, but a better solution for geometry of fire needs to be developed. One technique under consideration is a rifle mounted inertial measurement unit (IMU) to keep track of rifle orientation [1]. A tactical network would collect position and rifle orientation data from all friendly units and compute the offsets between the two [2]. If geometry of fires is conflicted, the network could then warn the rifleman of the issue and potentially prevent fratricide. To be of any use, however, this network requires having accurate positions of the individual riflemen in the unit.

B. PERSONAL NAVIGATION SYSTEM

As mentioned in the previous section, GPS does not provide the accuracy inside of buildings or onboard ships that is required for the task of de-conflicting friendly positions with rifle orientations. One method for obtaining a more accurate position estimate is the use of an inertial sensor-based personal navigation system (PNS). This approach uses a foot-mounted IMU to compute position and orientation of the individual through quaternions and double integration of acceleration data [3]. The benefit to using this approach is that it is self-contained, cost effective, and low weight when compared to other options.

The algorithm for position tracking is being refined in concurrent research to further reduce position error. At this point, the PNS testing only includes walking around a track, without intermediate stopping, and computing a position update with each step. The difference between the final and initial positions is then calculated. This method shows promise with accuracy within one meter [4]. In these tests, all motion was walking without the user stopping until reaching the destination.

There are several issues with the current PNS algorithm. First, the algorithm is viable only for walking. This means that running, sidestepping, shuffling, etc., cause inaccuracy in the position estimate. Another issue that arises when using IMUs for human tracking is sensor drift. During periods of long stops, the noise in the data causes the algorithm to compute movement when the person is in fact not moving. This is discussed in further detail in the background section of this thesis. Finally, another issue with this inertial tracking system is that the algorithm detects when a step is taken and then computes position and orientation from step to step. The problem with this method is that the user may be stationary, yet moving his legs in a manner that mimics walking. This phenomenon, especially when combined with the sensor drift, can cause large errors to occur. For example, if someone was lying face down on the ground and moving his legs, the algorithm could detect movement and therefore would erroneously update his position and orientation. A solution to these last two problems is the focus of this work.

C. THESIS OBJECTIVE

Development of an algorithm that can compute position based on running, sidestepping, crawling, etc., is not the goal of this research. The goal is to know when a person is performing these motions, which offers the ability for the PNS to switch between states and improve the accuracy of the position calculation. Posture estimation will allow the PNS to switch between a walking algorithm when the user is walking, a running algorithm when the user is running, and force the position to remain unchanged during periods when the user is stationary. Since this project is designed for use on the individual rifleman in the tactical environment, there is a desire to use the same inertial devices that are already in use for both position estimation and rifle orientation. This not only reduces the cost of the overall system but also allows greater compatibility throughout.

The remainder of this thesis addresses how a low-cost inertial device can be implemented to determine the user's posture and how this posture estimation can be used to improve the accuracy of the position-tracking algorithm. The goal of this research is to design a system that uses inertial sensors to track human posture over a range of possible states and to integrate the posture detection system with the current walking algorithm for position tracking. Furthermore, the optimal system will work in real time to allow the PNS to switch between states as the transition from one activity to another occurs.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

Previous work on posture estimation is examined in this chapter. The assumptions that were made throughout the research and the reasons why the assumptions are valid are stated. The reasons for choosing IMUs for posture tracking vice other technologies available on the market are discussed. A common understanding of the relevant technologies prior to discussing the posture-tracking algorithm and its implementation is also established.

A. ASSUMPTIONS

Assumptions should be made only when they are necessary to continue research. Several assumptions were made at the beginning of this research in order to start the development process.

The first assumption was that algorithms can be developed that are able to track motion other than walking. The current position-tracking algorithm works well when the user is constantly walking. If the user is sidestepping or running, this method for computing position does not work. It has to be assumed that another method for tracking these motions will be developed in the future. This assumption allows the posture-tracking algorithm to compute the state of running with the reasonable expectation that a running algorithm will be implemented in the future.

Second, PNS error grows over time due to the natural drift in the inertial sensors. It also only computes the position of the user in reference to a starting point and not the world coordinate system. This leads to problems when a combat unit moves via some other type of method like a vehicle. The second assumption is that GPS will be integrated into the personal navigation system as long as there is sufficient satellite coverage, and GPS accuracy can be assured. Integration with GPS allows the network to compute a latitude and longitude for the starting position of the individual in the world reference frame, thereby aligning everyone in the unit prior to any CQC type operations.

The final assumption made for this research limits the types of posture states being calculated. This assumption had to be made because the number of different human

poses is only limited by the imagination of that individual. The purpose of this thesis was not to track every pose that a human can create but to estimate those poses relevant to tactical movement patterns in combat units. Ultimately, the goal was to find the basic poses that the individual rifleman would use to move around the battlespace.

Before going further, it is necessary to explain the concept of posture. Posture, as it relates to this research, is a category of movement, or states. The states considered in this work are walking/standing, kneeling, prone, and running. The reasons for these choices are explained in subsequent chapters.

B. PREVIOUS WORK ON HUMAN POSTURE TRACKING

This research is not the first to use inertial sensors for posture estimation. In one study, inertial sensors were mounted on each limb of the body to track orientation of the limbs [5]. Optical sensors then were used for an offline learning phase, where data from the inertial sensors were linked to movement patterns observed with the optical sensors. This method allowed the person wearing the sensors to utilize a series of postures and the system to learn those patterns. When the learning phase was complete, the user was able to maneuver through an environment while orientation and position were tracked [5]. This process produced both posture and position estimations unique to each individual. The same application, however, poses challenges for combat units because each person needs to undergo the learning phase before employment on the battlefield. While this is possible, it requires technical support personnel who are trained in calibration of the system. This technique is also time consuming because of the need to update the algorithms for each individual as sensors are replaced or other modifications are added. What is desired is a system that is invariant to limb lengths and can be employed by anyone wearing it. This ideally allows the sensor system to be universal.

Another application of using inertial sensors to accomplish posture estimation comes from the medical field. Wenk and Frese [6] proposed that if a person's posture can be determined, it would be possible to analyze how long people were in ergonomically compromised positions during the workday. This data can be used to identify certain postures that may be linked to occupational diseases. Like the previous application, this

also uses a network of sensors attached to the body and contains multiple sensors on each limb.

Another application to the medical field is to use inertial sensors to track movement patterns, particularly in patients with Parkinson's disease [7]. By using inertial sensors to reconstruct movement patterns, it is possible to identify when certain tremor or seizure episodes occur in these patients. A human model could be built, and then observed movement patterns could be compared to a database containing both normal and abnormal movement patterns by a trained healthcare professional. The determination of the occurrence and severity of Parkinson's disease-related tremors could then be detected remotely [7]. Again, the drawback of this approach, when applied to the problem of posture tracking for combat personnel, is that the system requires multiple sensors to be worn on each limb and the construction of a full-body model before posture can be classified.

In another study [8], magnetic, angular rate, and gravity (MARG) sensors were used to create and simulate a human avatar that mimics movement in a virtual environment. As of 2004, only the right arm had been modeled. The MARG project evolved in 2005 when inertial sensors were used to analyze walking by modeling human gait cycles [9]. The goal of the MARG project was to get an exact model of the human and all of its limbs for use in simulation.

The four approaches described use multiple sensors on each limb of the body to accomplish the task of posture tracking, which increases the cost and weight of the system. To the extent possible, both cost and weight should be minimized to have the least impact on the combat unit. The current system that has been developed in this research only uses three sensors to accomplish the task of posture tracking. In addition, the desired system for tracking postures of the individual rifleman cannot use any type of visual motion capture device like a camera. Including such a device would be an additional component that would have to be mobile. This presents additional complications when working in a tactical environment if each rifleman has a motion capture device mounted to him.

C. SUMMARY OF BACKGROUND MATERIAL

In this chapter, assumptions were discussed as well as the justifications for each assumption. Posture was explained in the scope of this research, and some examples of previous work were discussed on the issue of posture estimation. In this thesis, the primary concern is creating a posture-tracking algorithm that augments the PNS for use in the tactical environment by combat personnel. What distinguishes this work from previous research is three-fold. First, this study seeks a generalized approach to posture estimation using an algorithm that does not require in-field calibration to the individual rifleman. Second, this study develops posture estimation with a minimal set of sensors. Finally, the posture estimation goal of this research requires no post-processing in order to deliver a real-time estimate of the user's posture.

III. POSTURE-TRACKING SYSTEM DESIGN

How the initial design of the posture-tracking algorithm came to fruition is explained in this chapter, and what states were selected and the reasons for their selection are explained. Furthermore, how the tracking algorithm evolved throughout the research is discussed and how the algorithm was tested in the lab, including the real-time implementation, is covered.

A. DETERMINATION OF KEY POSTURES

In the tactical environment, the rifleman deliberately keeps his weapon system oriented toward the enemy, or in the direction of immediate danger. Varieties of movements are performed to accomplish this task, to include walking, running, sidestepping, and crawling. There are also four basic shooting positions defined by the Marine Corps rifle marksmanship manual: standing, kneeling, sitting and prone [10]. These shooting postures offer the most stable platform to ensure accuracy when shooting. The focus of effort was to identify three of the basic shooting positions and include a differentiation between walking and running.

The sensor used for this research is the YEI data-logging sensor, which is composed of three axial gyroscopes, accelerometers, and magnetometers. The YEI sensor output is selectable to be raw or filtered data. The sensor is also capable of using internal algorithms and filters to produce quaternions and Euler angles directly [11]. Choosing the best approach was critical for this application. Early in the research, it was identified that the states of interest could be tracked by using only three sensors: one on the upper torso and one on each leg. The problem then became to determine what data could be used effectively for posture estimation. One thought was that by using the acceleration in the vertical direction, a relative height between the sensors could be calculated that would allow posture to be extracted. Using the data this way is imprecise, and an alternate technique was required. Another approach was to use the concept of transformation matrices to compute the relative positions between each of the sensor's origins during motion. Again, this approach was flawed because an accurate estimate of the starting

distances between sensors needs to be obtained prior to using the system. The system needed to be more robust for two reasons. First, it should work for anyone who wears the system without needing recalibration. In other words, the limb length should not affect the system's ability to track posture. Second, the exact mounting position of the sensors should not affect performance. The exact mounting position should be inconsequential as long as the lower appendages and the torso are monitored. The solution used in this design is to have the sensors detect the angle of the torso and legs with respect to the inertial frame and use these angles to determine posture. Given the nature of the components within the sensor, and its data output capability, this approach seemed to be the best option for this research.

B. TRANSITIONS BETWEEN POSTURES

The first task was to define an initial condition from which other postures could be derived. Standing seemed to be the logical choice for this because the person will be standing up when the system is turned on. The next task is to answer the question of how a human goes between states of standing, kneeling, and prone. Before this question is answered, a discussion of the different sensors locations and relative angles is needed.

1. Initial Orientation

To start the discussion of orientation of an object, it is first necessary to assign an initial coordinate system. The origin of the initial coordinate system is arbitrary, but it is fixed to a certain point and the coordinate axes do not rotate. The position of the origin along with the orientation of axes describes the coordinate frame. Since this initial frame is not moving, it is said to be an inertial reference frame. To describe an object with respect to this initial frame, two pieces of information are required. The first piece of information is the location of the object with respect to the origin of the fixed frame. To describe the position of an object, it is first necessary to pick a point on the object, which could be a centroid, center of mass, etc. Once the point is chosen, the location of the point can be described in the inertial frame. The second piece of information required is the orientation of the object. To describe the orientation, another coordinate system is attached to the object. The origin of the second coordinate system is fixed to the centroid

of the object. The position and orientation of the object's centroid are referred to as its reference frame.

It is often necessary to describe motion of an object with respect to a universal reference frame. In navigation, the north east down (NED) reference frame is commonly used. In the NED frame, the X-axis points toward Earth's north pole, the Y-axis points east, and the Z-axis points down and is perpendicular to the X and Y axes. For posture tracking, it is not necessary to start with the NED reference frame but an inertial frame that is defined when the system is first turned on. The human body has its own reference frame, which is referred to as the body frame. The orientation of the body frame for this research is defined as the X-axis pointing to the front of the body, the Y-axis to the right side of the body, and the Z-axis pointed down to the ground. When the posture-tracking system is first turned on, an inertial frame is created that has the same orientation as the body frame. As the body moves in space, the body frame's orientation changes, but the inertial frame is stationary. This allows the orientation and position of the body to be described in reference to this arbitrary inertial frame.

Each body part of interest is also assigned its own coordinate system. In this research, the chest, left leg, and right leg are assigned their own reference frame. Additionally, each sensor has its own coordinate system and reference frame. It is desired to attach a sensor to each body part in a way that aligns the sensor's reference frame and the body part's reference frame. Once the two reference frames are aligned, the sensor can be used to measure the rotation of the body part where it is attached. Since it is impossible to align the sensors' reference frame to the body parts' reference frame exactly, the sensors reference frame is set to the body parts reference frame in the program by adjusting the orientation of the sensor to align the two. For the rest of this thesis, it is assumed that the reference frame of any certain body part and the reference frame of the sensor attached to that body part coincide. For example, the centroid of the chest is not located where the chest sensor is mounted; however, the relationship between the two frames is known, which allows the chest orientation to be computed from chest sensor data. Since the pitch angle is used in this algorithm, the system is not user specific because the pitch for the chest and the sensor is the same.

The reference frames of interest in this research are the inertial frame, which is the body frame at time zero, the chest frame, the left leg frame, and the right leg frame. To describe how one frame changes orientation with respect to another, we use the concept of Euler angles. The orientation of a reference frame at a given time can be described by a rotation about the Z-axis by a yaw angle, the Y-axis by a pitch angle, and the X-axis by a roll angle [12]. When the posture-tracking system is turned on, it is assumed that the human is standing. When standing, the coordinate frame of each body part is aligned to the inertial frame and the yaw, pitch, and roll angles of each sensor are all zero. If the yaw, pitch, and roll angles of the sensor are not zero, as in the case where the sensors' mounting is not perfectly flat against the body, they can be set to zero by subtracting the initial angle from all the subsequent angles. A depiction of how the sensors are attached to the body and how each sensor is oriented with respect to the inertial frame is shown in Figure 1.

Figure 1. Sensor Mounting Locations along with the Basis Vectors of the Chest, Left Leg, Right Leg, and Inertial Reference Frames

2. Computing Euler Angles

With the starting orientation understood, the left leg sensor is used to describe how the kneeling position is realized. When standing, the left leg sensor's X-axis is aligned with the inertial frame X-axis. When the individual transitions from the standing state to the kneeling state, the knee bends and the left leg rotates about its Y-axis by an angle known as pitch. Nominally, standing is described as a pitch of zero degrees and kneeling, when the left leg is flat against the ground, has a pitch angle of -90 degrees. The YEI sensor is capable of computing the roll, pitch, and yaw angles directly, which makes the computation of pitch easily obtainable.

3. Kneeling and Prone

With the computation of Euler angles, it is possible to describe how each sensor rotates in its X-Z plane. The angle that describes this motion is pitch. Each sensor has its own pitch angle that describes its movement in reference to the local inertial frame. To detect kneeling with the left leg down, the pitch from the left leg sensor is read and compared to an angular threshold. If the magnitude of the pitch angle is below the threshold, then the posture is standing. If the magnitude of the pitch angle is above the threshold, then the posture is kneeling. The same technique is used on the right leg for cases where the individual is kneeling with his right knee on the ground. Putting these together, kneeling can be determined if one leg or both legs are above a pitch angle threshold.

To determine the threshold for kneeling, a series of experiments were performed to observe the pitch angles that corresponded to the movement. As expected, the pitch angle observed for standing was near zero, and the pitch angle observed during kneeling was close to -90 degrees. The algorithm, in its current state of development, works well if the user is only transitioning between the two states standing and kneeling; however, during periods of walking and running, the leg also rotates, causing changes in pitch angle. To reject any false kneeling readings that may arise during these two activities, another set of experiments was performed to determine the maximum pitch angles the leg sensors detected during walking

and running. With this data, the angular threshold was determined, and the algorithm can be updated with a reasonable value for the angular threshold.

Now that the transition between standing and kneeling postures has been explained, the logic for transitioning to the prone posture is discussed. To start, we examine what prone looks like. Prone is lying down with the body flat against the ground. It is essentially the same position as standing, but the entire body is pitched -90 degrees. Furthermore, to move from standing to prone for a human, kneeling must be used as an intermediate state. With this logic, kneeling must be detected before the chest pitch angle is taken into consideration. The detection of the transition between kneeling and prone is accomplished by first detecting the kneeling posture, then reading the magnitude of the chest sensors pitch angle and comparing it to an angular threshold. Much like the determination of the leg pitch threshold for kneeling, a set of experiments was performed to determine the optimal threshold value for the chest sensors pitch angle. The final threshold values for the kneeling and prone postures are 50 and 60 degrees, respectively.

4. Running

Up to this point, the states that the algorithm can detect are kneeling, prone, and upright postures, where the upright postures are standing, walking, and running. In the following, how running may be differentiated from standing and walking is discussed. Before the discussion on how running is detected, it is necessary to discuss some of the physical properties of running. The simple approach of determining Euler angles does not work for running. An alternate approach is needed for detection of this posture state. First, running can only be accomplished when the algorithm detects that the user is standing. This is not to say that the human cannot bend over at the waist and still be running, but simply that they cannot be kneeling or prone. To detect running, Z-axis acceleration data from the chest sensor is used.

For both walking and running, the leg motion can be described as a transition between two phases: a stance phase and a swing phase [3]. In the swing phase, where the leg swings forward, the majority of acceleration is in the X-axis. In the stance phase,

where the foot is in contact with the ground, the majority of the acceleration is in the Z-axis. Furthermore, during the initial part of the stance phase, when the foot strikes the ground, there is a large spike in the downward acceleration that can be easily detected. The YEI sensor measures the acceleration and displays it in terms of g-force, where 1g=9.80665 m/s^2. The sensor used in this research is capable of detecting up to 8g, which is adequate for this application since running produces approximately 7.0-7.5g.

Along with the detection of peaks in acceleration data, the frequency of these peaks must also be computed to describe running accurately. When running, humans usually run at a pace of 120 to 180 steps per minute [3]. This means that a step occurs at a frequency of two to three Hz. The downward force from the impact of the foot on the ground offers an opportunity to monitor the frequency of acceleration peaks and determine if it is over a certain frequency threshold. Two thresholds are used to detect the running state. The first is an acceleration threshold that determines if the foot strike had enough force to cause a peak to occur. This is important to differentiate the force felt during walking motions, which typically has a much smaller vertical force. The second is a frequency threshold. This is used to determine if a peak was detected due to a single event, like a jump, or if it is a periodic occurrence that corresponds to running. The frequency of running is between two and three Hz, which corresponds to approximately 30 samples for the sensor that was used. The posture-tracking algorithm detects all peaks in acceleration that are above a certain acceleration threshold. The algorithm then uses the previous 30 data points to determine if the acceleration is periodic at the appropriate frequency for running.

5. The Complete Posture-tracking Algorithm

The postures of running, standing/walking, kneeling, and prone have been described, as have the criteria for transitioning between them. It is important to point out that this algorithm does not differentiate between periods of standing and walking. These are considered the same state for purposes of integration with the PNS, which is discussed in the next chapter. A flow chart that encompasses the posture-tracking

algorithm discussed in this section is shown in Figure 2. The logic for how each posture is detected as well as the interconnection between states is shown

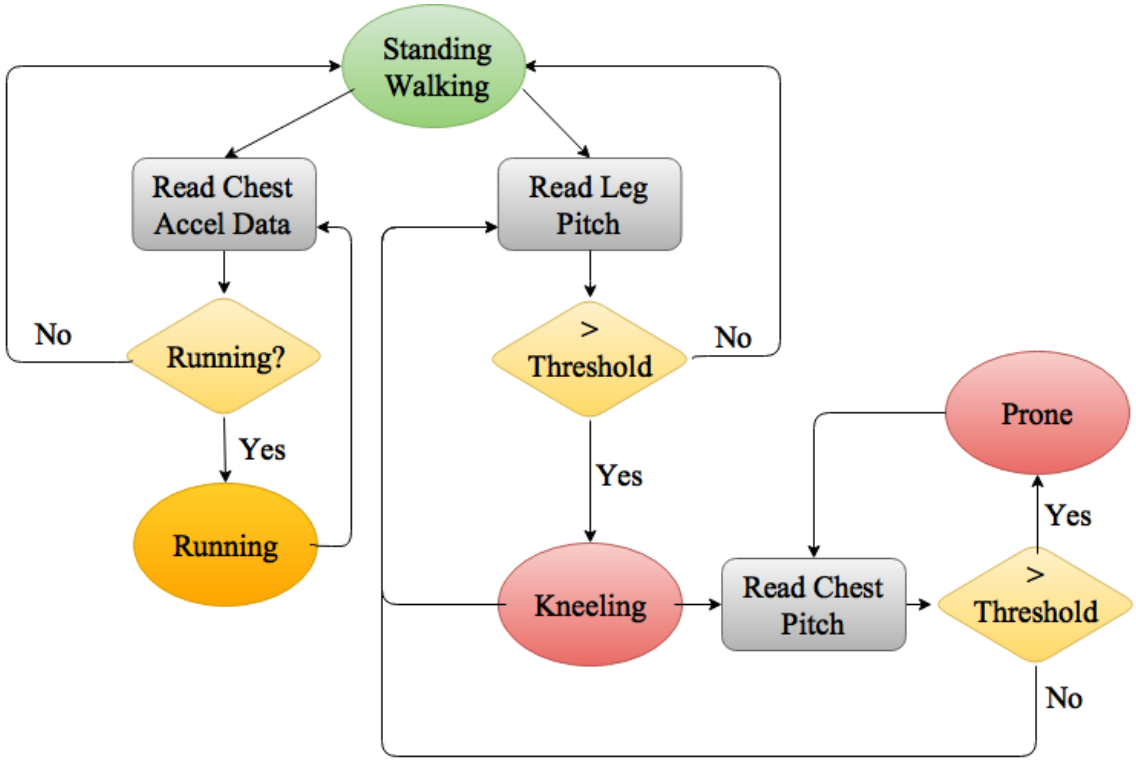

Figure 2. Flow Chart Showing Detectable States of the Posture-Tracking Algorithm

C. MODIFYING THE DESIGN

An initial design was implemented that was capable of detecting all of the postures that were discussed in Section B of this chapter. This design was then tested in the laboratory environment, and it was discovered that modifications were required for several reasons. Sensor location, filtering, and, finally, hysteresis control, are discussed in this section.

1. Leg Sensor Placement

The first design modification was the placement of the leg sensors. Originally, the leg sensors were placed on the feet. The thought behind this was that the PNS inertial

sensors would be mounted on the feet, and by collocating them, the same sensor could be used to accomplish both tasks. The issue that arose was when walking or running, the foot goes through a higher pitch angle than the rest of the leg. This became problematic for strictly using pitch angles of the legs to classify the kneeling state. After some thought, the sensor was mounted on the lower shin for the posture-tracking system. This allowed tracking of the lower leg as it hinges about the knee without the additional rotation about the ankle. This solution allowed the algorithm to have fewer erroneous kneeling detections. Of note, the PNS sensor remained attached to the foot throughout the research and testing, which adds more components to the complete system.

2. Factored Quaternion Algorithm and Low Pass Filtering

Another problem discovered during initial testing was that the data was noisy. There were two causes of the noise in the data. The first cause is from the sensor's internal algorithm for computing Euler angles. As mentioned, the YEI sensor contains three types of measurement devices: accelerometers, gyroscopes, and magnetometers for each axis. Internally, the sensor weights and filters each of these measurements based on the readings it records and then calculates the Euler angles. The gyroscopes can sometimes be erratic, which causes jumps in the data. An alternate approach is needed to obtain the Euler angle data. By using a factored quaternion algorithm (FQA), it was possible to use a combination of accelerometer and magnetometer data to obtain a quaternion to describe rotations of the sensors [13]. The FQA works by finding the direction of the gravity vector in the sensors. Once the gravity vector is computed, the angle between each of the sensors primary axes and this vector can be calculated. This angle is then used to find the roll and pitch quaternions. For the heading quaternion, the magnetometer data is used. Once all three of these quaternions are found, they are then combined to form a single quaternion that describes the rotation of the object. The FQA was implemented within the software code and can be found in the Appendix.

The calculation of Euler angles is done in two steps. First, the rotation matrix is extracted from the quaternion, and then the Euler angles are extracted from the rotation matrix. Eliminating the gyroscopes from the calculation greatly reduces the noise in the

pitch data. A low-pass filter was used to further remove noise from the angle and chest acceleration measurements.

3. Hysteresis Control

With the placement of the leg sensors on the shins and the implementation of both the FQA and low-pass filter, the system was again tested in the laboratory to determine the extent of the improvements. The posture-tracking algorithm was improved, showing less false positives for the kneeling, prone and a large reduction in error for the running state. The improvements are discussed further in Section D of this chapter. There is another problem with the algorithm that can be most easily understood by examining the prone state. To enter the prone state, the user must first be in the kneeling state. If the user were to lie down in the prone and then raise their feet toward their head, the leg sensors would read a pitch angle that was below the threshold for kneeling. This would cause the algorithm to read a standing posture, making the algorithm jump between standing and prone states. Because of this occurrence, a hysteresis control is needed. Using the prone state again as an example, we can describe the implementation of the hysteresis control. To transition from the kneeling state to the prone state, the system compares the magnitude of the chest pitch angle to a certain threshold. If the pitch is greater than the threshold value, then the state is switched to prone. Once in the prone state, the angular threshold is lowered, and all chest pitch readings are compared to this new threshold. Only when the chest pitch angle falls below this new threshold does the system transition from prone to another state. Once a transition from the prone state occurs, the angular threshold returns to the original value. To clarify this point, the following example is used along with Figure 3. The actual value of the chest angle threshold is 60 degrees. If the system detects a kneeling state and the absolute value of the chest pitch angle is greater than 60 degrees, then the output of the system transitions to the prone state. Once in the prone state, the absolute value of the chest angle must decrease to a value less than 40 degrees in order for the output of the system to transition from prone to another state. The same type of control was also implemented for the kneeling and running states.

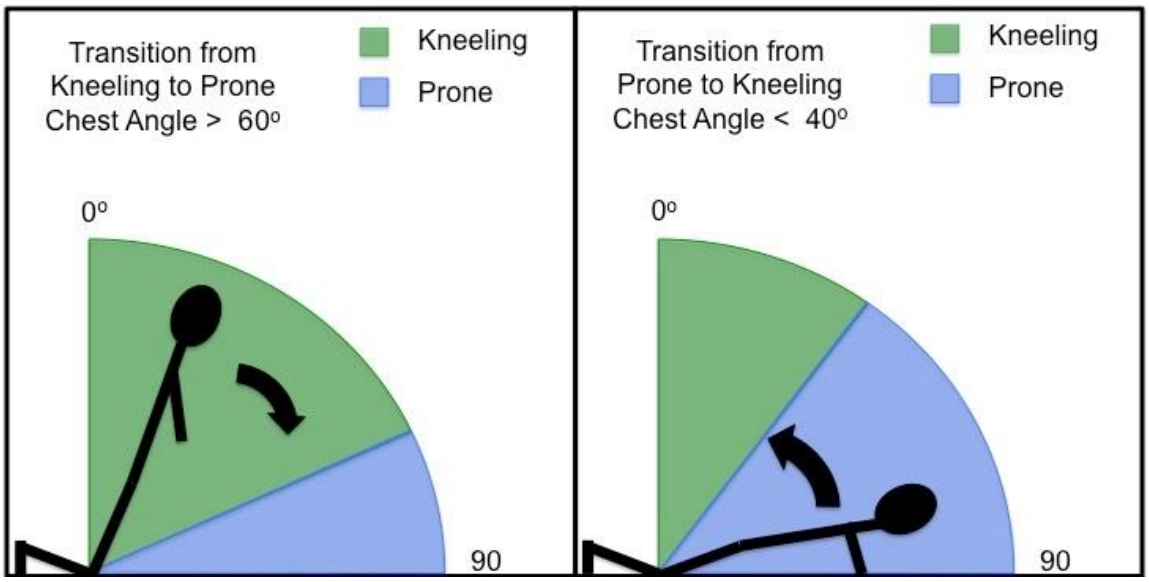

Figure 3. Visual Depiction of Hysteresis Control Concept for Transitioning between Kneeling and Prone Positions

D. RESULTS FROM LAB TESTS

So far, every experiment in the laboratory environment has been performed to get the posture-tracking algorithm to operate as designed. At this point, it is important to show the results of some of the modifications that have just been discussed. In Figure 4, there are three plots corresponding to the chest, left leg, and right leg sensors displayed from top to bottom. In each subplot, the green lines correspond to the pitch angles, in degrees, that were obtained without the use of the FQA, low-pass filter, or hysteresis control. Each of the vertical gridlines represents 100 degrees. The black lines correspond to the posture-tracking algorithm's output. The states are displayed along the Y-axis. The state at the top of the Y-axis is running, the next state down is standing/walking, the next state down is kneeling, and the state at the bottom of the Y-axis is prone. The X-axis in each subplot is time. Particularly of interest in this plot is between the times of approximately 65 to 85 seconds. This region of the plot shows many instances of detecting the running state when in fact the user was walking. Another region of the plot that is of particular interest is at approximately 50 to 63 seconds. During this period, the algorithm detected a transition from prone to standing, then back to prone. All other

portions of the plot were verified to be true representations of the movements performed. In contrast, a plot of the same data with the FQA, low-pass filter, and hysteresis control implemented in the system is shown in Figure 5. Examining the first region between 65 and 85 seconds, we see that the erroneous running states have been corrected. The FQA and low-pass filtering are what allowed this correction to take place. Similarly, the second region of interest between 50 and 63 seconds shows the corrections made by the hysteresis control. In Figure 4, the false kneeling state was created by lying flat on the ground and rotating the lower legs forward toward the head. With the hysteresis control in place, the same region in Figure 5 displays a constant prone state for the same time period. In both figures, the time periods of interest have been circled in red.

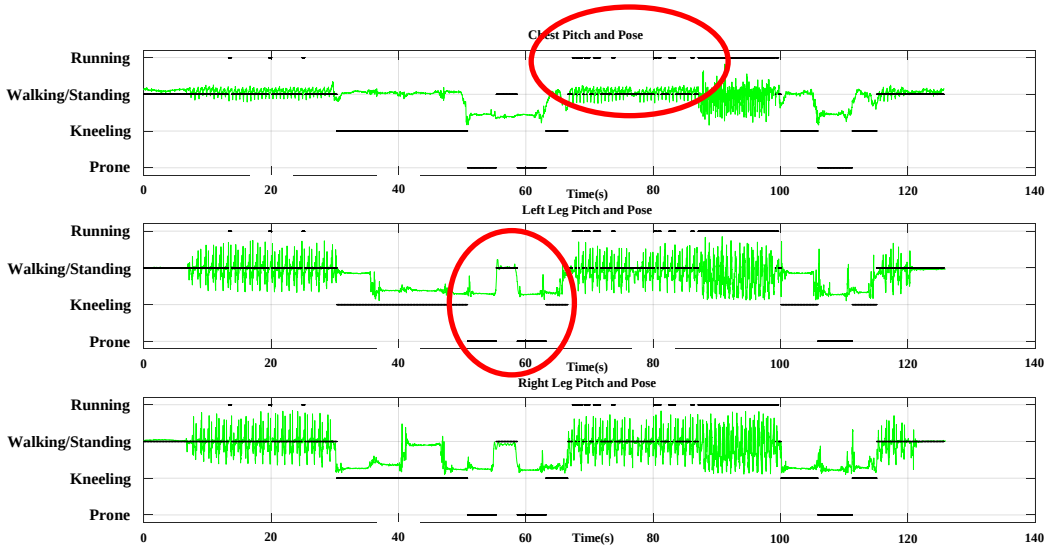

Figure 4. Posture-Tracking Algorithm Results before FQA, Low-Pass Filter, and Hysteresis Control Were Implemented

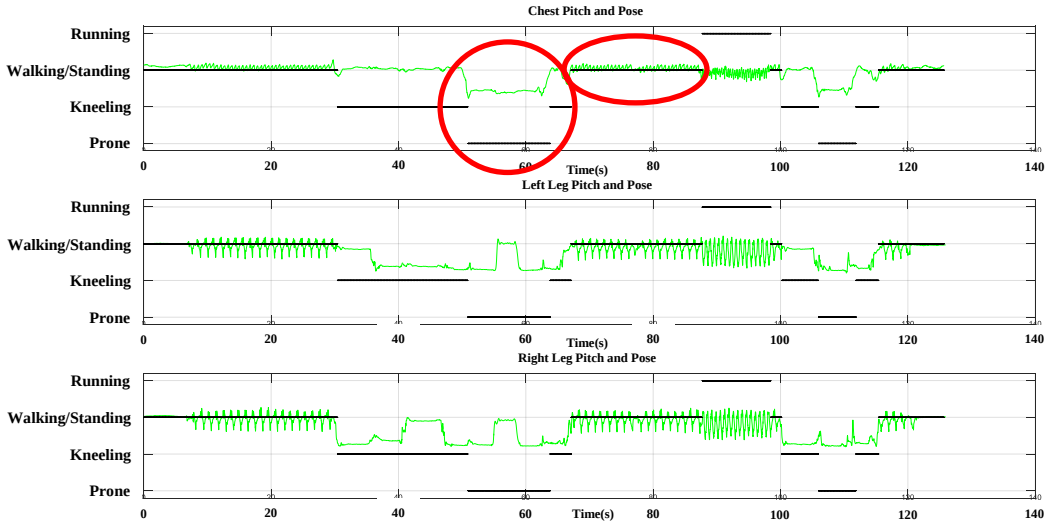

Figure 5. Posture-Tracking Algorithm Results after FQA, Low-Pass Filter, and Hysteresis Control Were Implemented

E. REAL-TIME TRACKING

The YEI sensor used for this research saves the collected data to a text file on the internal storage of the device so that it can be easily transferred for post-processing. Up to this point, all data was collected and saved on the sensor's internal memory and then transferred to a computer where post-processing was performed. This method made sense for the initial design phase so that items such as angular thresholds, acceleration thresholds, and filter parameters could be adjusted in post-processing easily. In addition, the PNS does not yet function in real time, so integration of the two systems was most easily done with post-processing. Since one of the goals of this research was integrating these systems, they needed to use the same techniques for processing.

The need for real-time tracking of position goes back to the original problem that the PNS was trying to solve. An accurate position estimate in a GPS denied environment is required for the geometry of fire tracking algorithm to operate correctly. This requires a real-time position to be supplied to the network so that friendly nodes can be compared to rifle orientations. Since the PNS needs to transition to real-time eventually, so does the posture-tracking algorithm.

Implementing the posture-tracking system in real time was not a difficult task once it was shown to work accurately with post-processing techniques. The posture-tracking algorithm only required slight modification, and the added complexity of the real-time system was due to interfacing the sensor and the software program. When the real-time system was tested, it performed as expected but did introduce another problem. In order to read the data, the sensor has to be physically connected to a computer via USB cables. This is problematic since there would need to be wires from each sensor connecting them to an onboard computer when worn by the rifleman. A solution to this problem was not pursued any further in this research, but one idea is to use sensors that are capable of sending the data wirelessly.

F. SUMMARY

How the posture-tracking algorithm has gone from concept to implementation was discussed in this chapter. A description of the running, walking/standing, kneeling and prone states was given, as well as the logic for transitioning from one state to another. Certain problems were identified in the initial design, which required it to be refined. Noise filtering techniques and hysteresis control were discussed, and application of these techniques was shown to make great improvements to the robustness of the system. Finally, the posture-tracking system was modified and tested in real time for use with future versions of the PNS.

IV. INTEGRATION OF THE POSTURE-TRACKING AND PERSONAL NAVIGATION SYSTEMS

With a working posture-tracking system, it is now time to test the system in more dynamic environments. One of the goals in this research is to integrate the posture-tracking system with the personal navigation system and analyze how accuracy of the PNS is improved. Testing of the system under different user conditions is briefly discussed in this chapter. In the remainder of the chapter, the test plan for the integrated system, the performance of the posture-tracking system, the method for integrating the PNS and posture-tracking system, and, finally, the results from the track testing is discussed.

A. INTEGRATED SYSTEM TESTING

1. User Variation

Up to this point in this research, there has been only one user for the posture-tracking system. It was necessary to do this so that the number of variables was reduced while the system was still in the design phase. Since one of the design goals was to have a system that performed well for a variety of users, it was necessary to have other people test the system. The system should be robust enough that when another person wears it, performance is not degraded. For testing purposes, the system was tested by another user, which had dissimilar body dimensions. The purpose of this test was to validate that limb length has no significant impact on system performance. The user was allowed to perform any sequence of walking, running, standing, kneeling, and prone as desired. Since there was not a certain pattern or sequence that was followed, a video was taken so the system output could be compared to the actual movements performed. The system was found to adequately represent movements around the laboratory for all states for both users. These results were necessary prior to integrating the PNS and posture-tracking system, since another individual will wear both systems.

2. Test Plan for System Integration

The PNS was previously tested by walking around a track where total distance traveled is known. The distance computed by the PNS was then compared to the actual distance traveled, and it was found that the PNS error could be less than 1.0 m [3]. To achieve this accuracy, the user walked around the track without stopping until the final destination was reached. If this were a realistic model of human motion, there would be little use for a posture-tracking system. As mentioned in the introduction chapter of this thesis, long periods where the human is motionless can lead to significant sensor drift. Uncorrected, this drift can lead to growing errors in the position computed by the PNS. The position error is further compounded if the user is stationary but continues to moves the legs in a manner that resembles the walking motion.

Since the PNS system was being tested by walking around a track, it made sense to test both systems in this environment. When testing the two systems, a plan was needed that could capture the accuracy of the PNS for walking without stopping. This would serve as a baseline against which to compare results of the integrated system. The test plan should also include periods where the human was motionless and periods where the user was not advancing forward but the legs are moving in a manner that emulated walking. This let the drift error build up in the sensors during periods where the person was motionless as well as creating false periods of translation in the position around the track. After some collaboration with the individual researching the PNS, a test plan was formulated that encompassed everything that was needed for testing both systems. It was decided that there would be four distinct laps around the track. The user would wear both systems consisting of the three posture-tracking sensors and four sensors mounted to the feet for position calculations. In the first and second lap, the user started on a predetermined point on the track and then walked, without stopping, in a loop around the track until returning to the starting point. This same procedure was followed in previous research on the PNS and gives a baseline of the accuracy of the PNS. The expected output of the posture-tracking system is that it would show the state walking/standing for the entire time.

In the next two laps, the user followed the same path as for the first two laps, but this time the user stopped every 100.0 m to perform certain maneuvers. The user started at the same starting point that was used for laps one and two and walked 100.0 m. At the 100-m mark, the user stopped moving and performed a kneeling pose for approximately 30 s. During this kneeling period, the user was free to shift his body, reposition his legs, and even simulate reaching down to tie his shoe. After this, the user returned to standing where he walked another 100.0 m. At the 200-m point, the user stopped again and performed a kneeling pose. This time it was performed with the opposite leg against the ground. While kneeling, the user performed similar activities to those performed during the first kneeling period. Once complete, the user returned to the standing posture and walked another 100.0 m. At the 300-m point, the individual transitioned to the prone position. In the prone position, the user shifted his weight from side to side, moved his feet in a walking manner, and even rolled onto to his back. After approximately 30 s in the prone position, the user transitioned to the standing posture and walked the final 100.0 m back to the starting position. The expected output for the posture-tracking system was accurate detection of each posture during the test.

The reasoning behind the maneuvers performed in laps three and four was to introduce error to the PNS. In the current form of the PNS, it is assumed that the individual moves in a certain manner. Once these assumptions are violated, the PNS position output will be less accurate. The idea is that once the posture data is integrated into the PNS, the periods where the user is motionless can be filtered out. This filtered data can be compared to the data from laps one and two to determine the improvement in accuracy made by the posture-tracking system.

Before the data from the two systems can be integrated, there is one additional consideration. In order to integrate the two systems, it is necessary to time-align the data. In the YEI sensor suite, the sensors internal clock can be set. To ensure that all seven sensors are synchronized, the same computer was used to set each sensor. Without time data, the integration is not possible without an alternate method of aligning data. In order to have an alternate method to align the data, a distinguishable jump was performed at the

beginning and end of each lap. If necessary, the acceleration data can be roughly aligned using the peak created by this jump.

B. POSTURE-TRACKING SYSTEM RESULTS FROM TRACK TEST

For laps one and two, the posture tracking system accurately detected the state of walking/standing for the entire duration. The only exception was during the jump at the beginning and the end of both laps, which was enough vertical acceleration to register as a brief period of running. The plot of posture data for each lap follows the same format as Figure 4 in the previous chapter. The output of the posture-tracking system for lap one of the track test is shown in Figure 6. The output matched what was expected, where the user was determined to be walking/standing the entire time. The circled areas highlight where a jump was performed at the beginning and the end of the lap. As shown, the system registered this jump as a very short period of running.

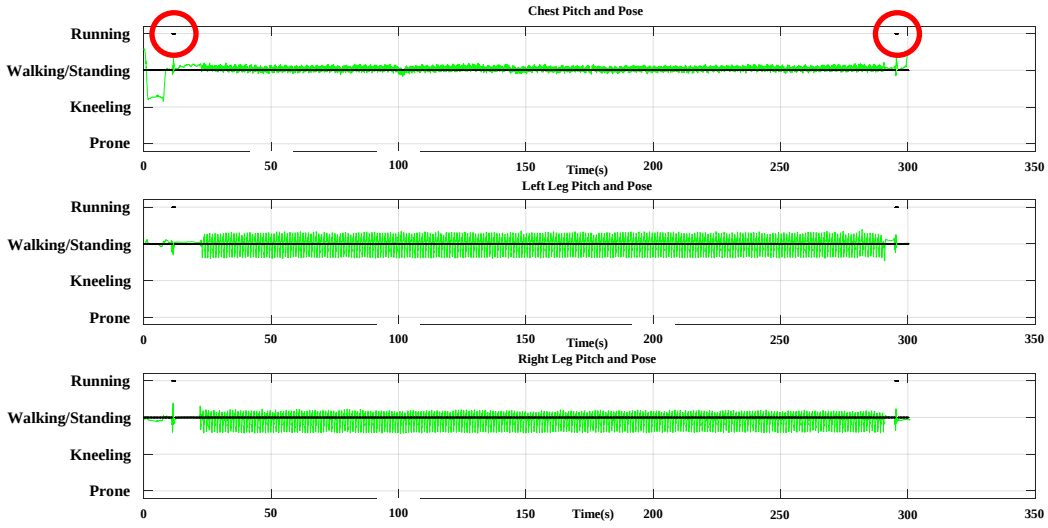

Figure 6. Posture-Tracking System Output from Lap One of the Track Test

The posture-tracking system output from lap two of the track test is shown in Figure 7. This plot has very similar results to the first lap of the track test. The posture-tracking system detected the user to be in the walking/standing state for the entire

duration of lap two, except for the jump that was performed at the beginning and end of the lap. Similar to lap one, this jump was interpreted as a short period of the running state and is again circled. In the first two laps, the PNS position estimate was unchanged by the posture data.

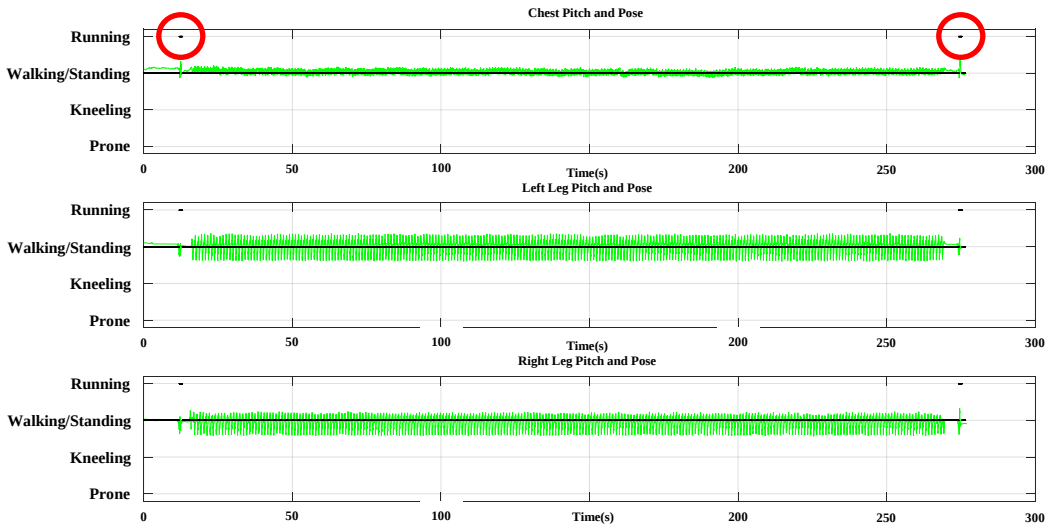

Figure 7. Posture-Tracking System Output from Lap Two of the Track Test

In laps three and four, a video was recorded so the posture-tracking system output could be compared to the actual maneuvers that were performed. Once the video was time-aligned to the data, it was found that the posture-tracking system was accurate for each of the states. Additionally, there was less than a 0.5-s delay when transitioning between any of the posture states. A plot of the posture-tracking systems output for lap three of the track test is shown in Figure 8. Similar to the first two laps, there is a momentary period at the beginning and end of the data where the system detected the running state. This is again due to the jump that was performed. Between approximately 15 to 70 s, the user was walking the first 100 m. At the 100-m mark, the user transitioned to the kneeling state, with the right leg flat against the ground, and remained there until approximately 105 s. Within this period of kneeling, there are two brief periods where the system detected the prone state. When compared to the video of this lap, these two

periods corresponded to the user leaning forward at the waist, causing the chest sensor to rotate. This rotation of the chest reference frame caused the system to detect the prone state. This result is consistent with the expected result. The next period, between 105 and 175 s, shows the user walking from the 100 to 200-m mark on the track. The system accurately detected the walking state for the duration of this period. At the 200-m point, the user again transitioned to the kneeling state, with the left leg now against the ground. The same kinds of small movements were performed while in the kneeling state, causing the system to briefly detect the prone state. Between 205 to 270 s, the user stood up from the kneeling position and walked to the 300-m mark. Upon arrival at the 300-m mark, the user then transitioned to the prone state through a brief period of kneeling, all of which can be seen on the plot. Between 270 and 320 s, the user remained in the prone state and performed a number of leg movements and repositions. The system detected the prone state for the entire period, showing that the hysteresis control was working. The last portion of the data correctly depicts the user walking the final 100-m portion of the lap back to the starting position.

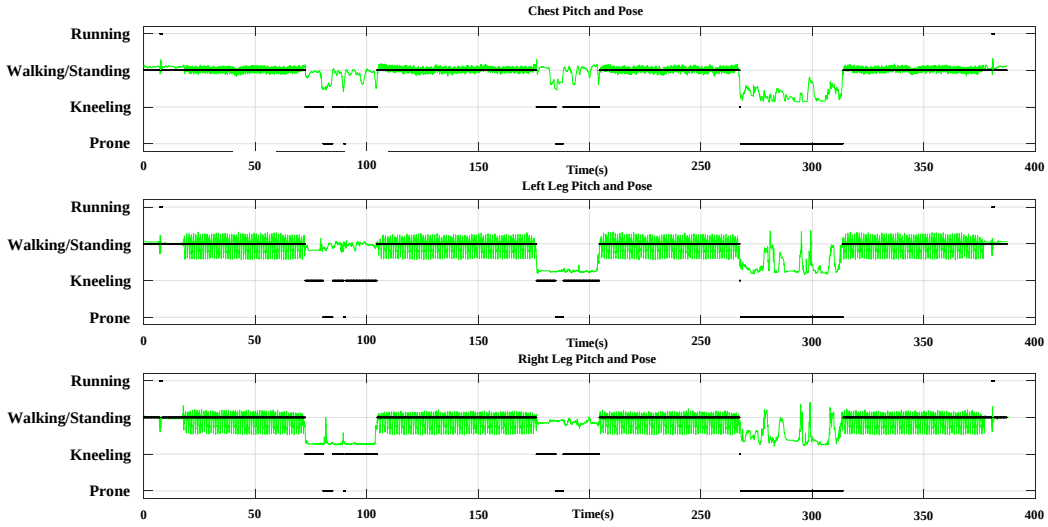

Figure 8. Posture-Tracking System Output from Lap Three of the Track Test

The fourth lap had a similar structure to the third lap and yielded similar results. There were some subtle differences in what movements were performed while kneeling. The plot of the posture-tracking system output during the fourth lap is shown in Figure 9. The results obtained are very similar to the results of lap three. The only difference between lap four and lap three is that in both periods of kneeling, the user switched which leg was against the ground and then switched back. For instance, at the 100-m mark, the user first had his right leg flat against the ground. The position of the legs was then switched so the left leg was against the ground. Finally, the position of the legs was switched back to the original configuration. The system continued to detect the kneeling state throughout each of these movements. Overall, the system performed as expected.

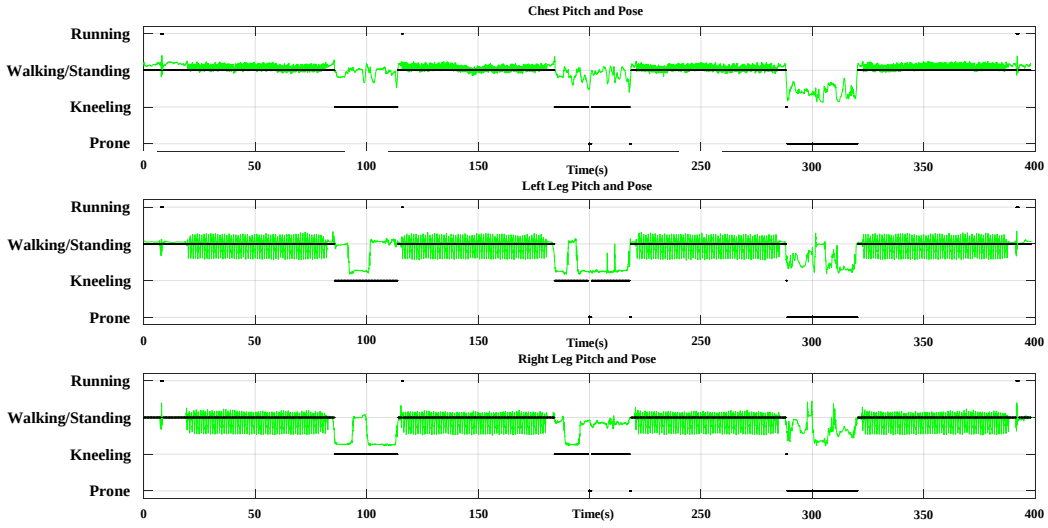

Figure 9. Posture-Tracking System Output from Lap Four of the Track Test

C. DATA INTEGRATION

The goal in the track test was to add purposefully created error into the PNS position estimate and then use the posture information to filter the errors. To do this, it is necessary to understand how the PNS computes position. The position estimate has two components. The first is the distance traveled by the individual in a single step. In the PNS algorithm, there are two phases of the gait cycle for walking. The first phase is

stance, which is the portion of a step where the foot is in contact with the ground. This phase starts when the individual's heel strikes the ground and ends when the toe leaves the ground to start the next step. The major assumption is the foot is stationary throughout the stance phase. The second phase is swing, which is the portion of a step where the foot is not in contact with the ground as it swings through the air. The PNS uses angular rate thresholds to determine whether the individual is in the stance or swing phase. When the foot is in the stance phase, no position updates are computed. When the foot is in the swing phase, the system computes the distance traveled by double integrating the acceleration data. The second component in the position estimate is the orientation. To compute the orientation of the individual, sensor data is used to compute a quaternion, which is then used to compute orientation during that step. It is important to note that the position is only updated when the individual is in the swing phase of the step. The orientation and distance traveled by the foot throughout each step is combined to produce the updated position estimate.

When the two systems are integrated, the posture-tracking system should only allow the position to be updated when the individual is walking. Furthermore, the position should be held constant when the individual is in a non-maneuvering state. For this reason, the posture states of kneeling and prone are combined to form the non-maneuvering states, and the walking, standing, and running states are combined to form the maneuvering states. When the posture-tracking system detects a maneuvering state, it allows for normal operation of the PNS. When the system detects a non-maneuvering state, it holds the PNS in the stance phase, which does not allow any position updates to occur. By aligning the data from the two systems in time, the periods where a non-maneuvering state occurs can be filtered out of the position data, thereby giving a more accurate estimate of position throughout each lap around the track.

For each lap in the track test, the user wore four PNS sensors attached to the foot. Each of these four sensors was processed separately for the purposes of position estimates. In the first two laps, the user was walking only. Since walking is a maneuvering state, the PNS was allowed to collect data normally throughout the entirety of each lap, and the position estimates were unchanged. On the third and fourth laps, the

individual made several transitions between maneuvering and non-maneuvering states. In these two laps, position was computed both with and without the posture data. Since there were four different sensors for the PNS, the two laps provided eight data sets to analyze. Figure 10 was produced from one of the four sensors on lap four. The red dots represent the user's position without the use of posture data, and the blue dots represent the position computed with the aid of the posture-tracking algorithm. In each case, a dot represents an individual step that was taken.

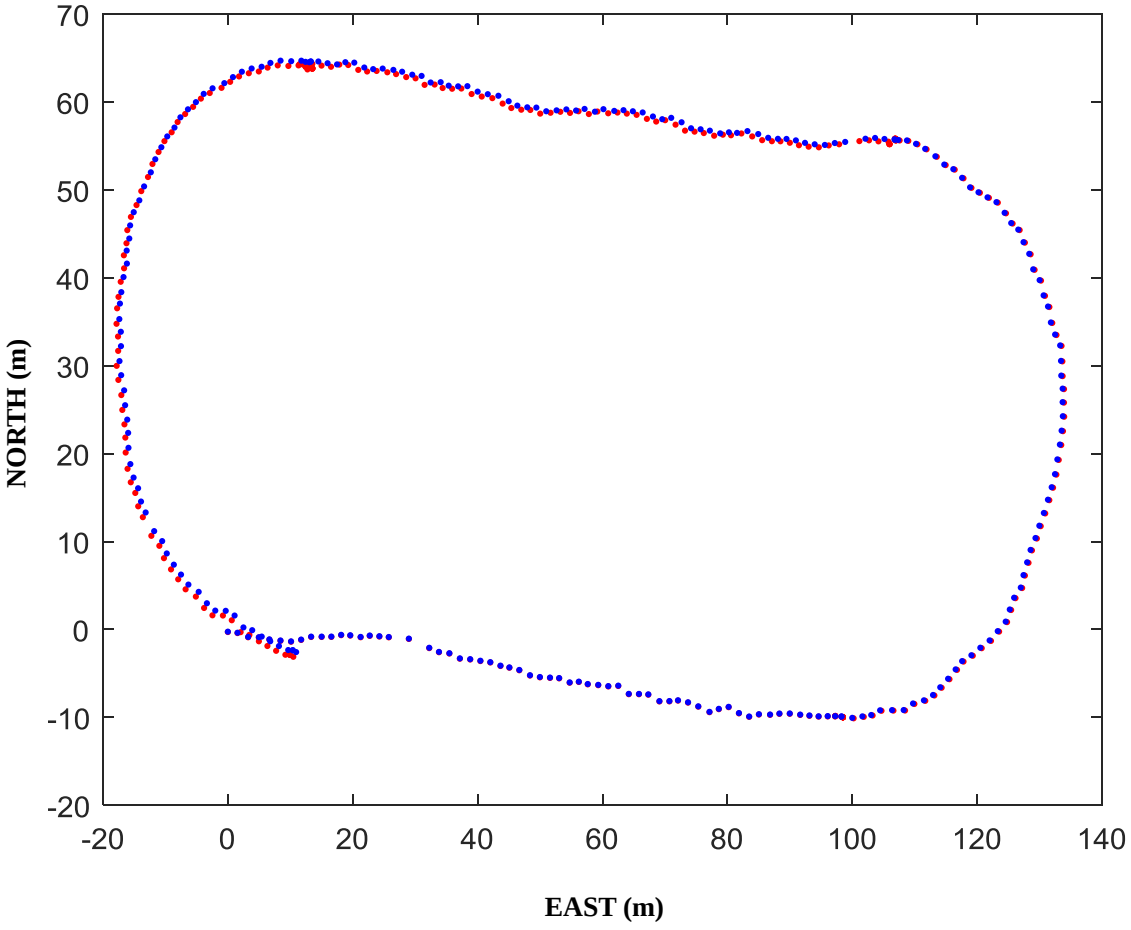

Figure 10. Overlay of X-Y Position from PNS without Posture Data (Red) and Position with Posture Data (Blue) for Lap Four, with PNS Sensor 2

During the first straight portion of the track, the position estimates coincide. At the end of the straight portion, just prior to the first left turn, the positions start to deviate

from one another. This point is where the first non-maneuvering period occurred. At the end of the first turn, just prior to the second straight portion, the second non-maneuvering pose was conducted. This caused further divergence of the two position estimates. Throughout the second straight portion, the position estimates are noticeably different. At the end of the second straight portion, just prior to the final left turn, the last non-maneuvering pose was conducted. This point on the plot shows the largest divergence in the two position estimates. This error is carried through the last turn until the final position is reached. The compounded error throughout the lap affects the total distance traveled and the distance from start-point to end-point. The leg movement during each non-maneuvering state caused the PNS, by itself, to detect false steps. This changed both displacement and orientation during those periods. Once filtered by the posture data, the position estimates are much smoother and had better results in terms of total distance traveled. Figure 11 is a plot of the same data displayed in Figure 10, zoomed in to show the 300-m mark, where the user was in the prone state. During this non-maneuvering portion, the PNS position estimate is updated numerous times, as indicated by the large number of steps detected in red. The points plotted in blue are the position estimates that were obtained from the PNS with the use of posture data. Comparing the two plots, we see that the posture data improves the accuracy of the position estimate during periods where the user is in a non-maneuvering state, creating a much smoother path.

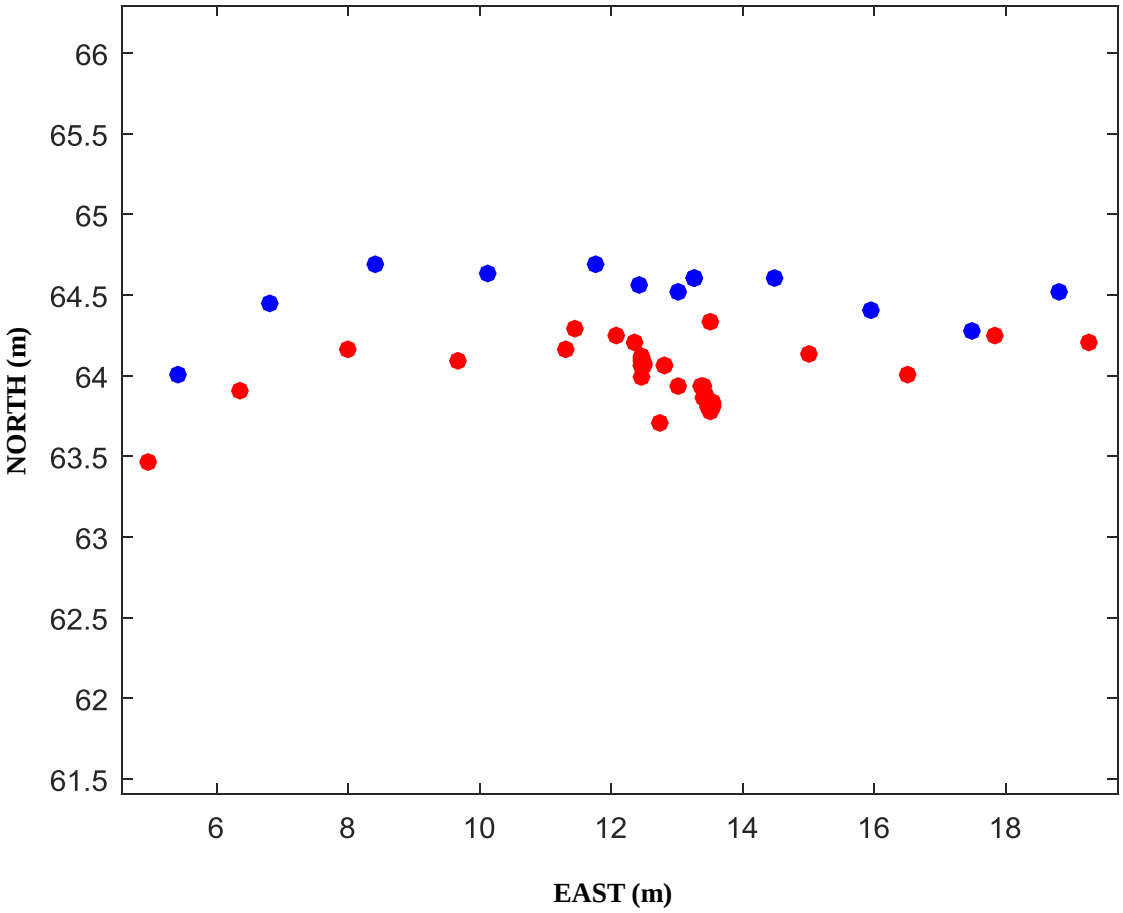

Figure 11. Overlay of X-Y Position from PNS without Posture Data (Red) and Position with Posture Data (Blue) Zoomed to the 300-m Mark on the Track

The data from each sensor was compared for both laps in the same way. In each case, the use of posture data significantly decreased error during periods of non-maneuvering. By doing this, we improved not only the error in total distance traveled but also improved the accuracy of the position estimate throughout each lap. Table 1 is a summary of the eight data sets that were analyzed. In each case, total distance traveled, distance from start point to end point, and total steps taken are compared between the PNS by itself, and the PNS output filtered with posture data. Each set of data is averaged and compared to the actual distances and steps taken. The actual values used to compute error are 400 m for total distance traveled, 253 steps taken, and zero m between start and end points.

Table 1. Results from the Track Testing for the Combined Personal Navigation and Posture-Tracking Systems

Lap	Sensor	PNS Alone			PNS with Posture Data		
		Total Distance (m)	Dist. from Start to End Point (m)	Total Steps (steps)	Total Distance (m)	Dist. from Start to End Point (m)	Total Steps (steps)
3	1	402.61	17.56	311	388.05	19.59	235
3	2	406.50	7.16	287	395.91	8.80	222
3	3	413.74	16.88	309	400.42	16.30	242
3	4	406.92	10.98	296	395.69	13.06	229
4	1	407.29	11.01	308	397.77	12.97	255
4	2	408.67	10.35	307	399.33	10.65	253
4	3	413.63	10.76	317	405.22	11.83	257
4	4	413.87	3.15	307	402.01	2.61	252
Average		409.15375	10.98125	305.25	398.05	11.97625	243.125
Error		9.15375	10.98125	52.25	1.95	11.97625	9.875

D. SUMMARY

In this chapter, a test plan was discussed that involved the PNS and posture-tracking system. The performance of the posture-tracking system was discussed for different users as well as the performance during the track testing. Finally, the results of system integration was discussed, and the accuracy of the position estimate was shown to be improved with the use of posture data to filter periods of non-maneuvering states from the position estimate. In the context of the bigger picture, the position obtained from the PNS will be used to compute friendly positions for the geometry of the fire-tracking algorithm. In this algorithm, it is important to have an accurate estimate of each rifleman's position at any given time. From the data in Table 1, it can be concluded that the error in total distance traveled and number of steps taken is significantly reduced when posture data is used. A more important result is that the position estimate throughout each lap is improved. In Figure 11, the position computed by the PNS alone during the non-maneuvering state shows a large variation, up to 10.0 m in one case, when it is known that the individual was not moving. It is desired that the position is accurate

for every time period and not just the total distance traveled. The distance from start point to end point is shown to be somewhat unchanged but is most likely due to inaccuracies of the starting position computed by the PNS during the beginning of each lap. The jump that was performed at the beginning of each lap could have caused the PNS sensors to become inaccurate at the beginning of the lap. Further testing with the PNS is required to determine the solution for these inaccuracies. The results of this chapter show the potential for using inertial navigation along with posture detection to obtain an accurate position estimate.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS AND RECOMMENDATIONS

A. SUMMARY

The modern battlefield is a complex and dynamic environment where tempo of operations must be maintained in order to be successful. One of the most challenging environments faced today is close quarters combat, where it is increasingly difficult to maintain awareness of friendly units. Previous research has shown the feasibility of creating a system for tracking geometry of fire, where friendly positions and rifle orientations are compared on a network. The rifleman can then receive warning if a potential fratricide issue exists. This system can increase confidence of friendly units as well as increase the situational awareness of the rifleman, thereby increasing tempo. In order to implement this system, friendly positions must be known accurately within the network. For this reason, a personal navigation system must be used that is capable of determining the location of each person in the unit.

Modern battles are often fought in locations where GPS is unreliable due to lack of sufficient satellite coverage. For this reason, GPS alone is not a viable solution for this application. An alternate method of determining friendly position is desired for use in this fratricide detection system. One potential solution is the use of a personal inertial navigation system to determine friendly force positions. Inertial based navigation does not come without its own difficulties. Current research shows promise that inertial based navigation techniques can produce the accuracy needed for the system but only under ideal testing conditions and numerous restrictions on the type of movements performed.

The current inertial based personal navigation system has shown accuracy of approximately 1.0 m over a 400-m tracking course. In order to achieve this, the user is only allowed to walk and cannot have any long periods of motionlessness. Any other motion causes the inertial sensors to drift and causes compounding errors in position estimates. To make the personal navigation system more robust, there must be a method of determining what type of movement the user is performing. The posture-tracking system developed in this research is capable of determining when the user is

maneuvering and when they are stationary. Knowledge of the user's posture can determine when to allow the navigation system to compute updates to the position estimate and when the position is to be held constant.

The goal of this research was to develop a low-cost, inertial based posture-tracking system to be integrated with the personal inertial navigation system that was the subject of concurrent research at the Naval Postgraduate School. The posture-tracking system that was developed is capable of detecting the difference between the running, walking/standing, kneeling, and prone states. After the development was complete, the integrated system was tested. The results of these tests show that the posture information can in fact improve the position estimate obtained by only allowing position updates when the user is moving. The integrated system was tested by walking in a closed loop around a track. It was found that the addition of posture data did not have a significant impact on the distance error between the start and end points during this type of test; however, the error in total distance traveled was reduced from 2.29% to 0.49% with the aid of posture data. The error in total steps taken was also reduced from 20.65% to 3.9%. The addition of posture data had the most impact during periods where the user was stationary, nearly eliminating false periods of translation. Overall, the addition of posture data creates a more accurate estimate of actual position compared to the PNS alone.

The original contribution in this research is a real-time posture-tracking system that is implemented without the use of any external motion capture devices and does not need to be calibrated to each individual. The improvements made to position estimates by the navigation system show an increasing potential for use in the fratricide detection application. As research of this inertial based personal navigation system expands to estimate position based on running and other translational movements, the posture-tracking system will be a vital part to determine when the system should use each algorithm.

B. FUTURE WORK

1. Real-time Position Estimates Using the PNS

The PNS is still in the initial phases of research, and several improvements are desired in order to integrate it with the geometry of fire tracking algorithm. The first of these improvements is to have the PNS compute position estimates in real time. This is necessary because the geometry of fire tracking algorithm requires position of friendly units to compare with rifle orientations.

2. PNS Algorithms for Running and Other Translational Movements

Another desired capability of the PNS is to be able to produce position estimates from other states besides walking. Ideally, the PNS would be able to have a running and sidestepping algorithm that could be used when that posture state was detected. These improvements to the PNS would make the system much more robust and more realistic for the combat environment.

3. Additional Posture States

The posture-tracking system that was implemented in this research focused on several basic posture states. In order to make the system more robust, there is a need for additional posture states. Additional states include sidestepping, jumping, climbing stairs, and separating standing from walking. These additional states are required in order for the future versions of the PNS to switch between algorithms for running, sidestepping, etc.

4. GPS and PNS Integration

As stated in Chapter II, the PNS only computes position in reference to some arbitrary starting position. In order to use position data in a system that computes geometry of fire conflicts, all user positions must be transformed to a common coordinate system. To do this, GPS has to be used in order to compute a starting point. To integrate the PNS and GPS, there needs to be a set of conditions for which the system switches to the PNS for position when the GPS becomes inaccurate. Conditions also need to be determined for when the system switches back to GPS for position when the GPS accuracy improves.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX

A. MATLAB CODE FOR POST-PROCESSING POSTURE-TRACKING ALGORITHM

The following code was used for post-processing all posture data that was collected throughout this research. This code requires the three data files from the sensors as well as the FQA and Euler functions. To ensure proper operation, the following data needs to be collected from the YEI sensor: date/time, microseconds, corrected gyroscope, corrected accelerometer, and corrected compass. These settings can be adjusted in the sensor's configuration file.

1. Main Posture-tracking Algorithm

```
% This program is then able to compute to a reasonable degree of
% accuracy the difference between walking/standing, running, kneeling,
% and prone postures.
% This program takes a sample of the first 40 data points of the
% sensors pitch data and calculates a reference angle.
% The data output from the sensor is acceleration and compass data
% which is then transformed with the FQA and Euler functions developed
% by Dr. Calusdian and Dr. Yun.

close all
clear all
clc

% Import and rename the data
% Type the name of the file that you would like to open here.
nameOfFile='11Sep15_4thlap_maneuver';


chest=importdata([nameOfFile,'_c.txt']);
chest.textdata=chest.textdata(2:end,:);   %Gets rid of header lines
left=importdata([nameOfFile,'_lf.txt']);
left.textdata=left.textdata(2:end,:);     %Gets rid of header lines
right=importdata([nameOfFile,'_rf.txt']);
right.textdata=right.textdata(2:end,:);   %Gets rid of header lines

% Dummy variables

hourc=[];
minutec=[];
secondsc=[];
EAc=[];
EAl=[];
```

```
EAr=[];
timec1=[];
timelf1=[];
timerf1=[];
rollc1=[];
pitchc1=[];
yawc1=[];
rolll1=[];
pitchl1=[];
yawl1=[];
rollr1=[];
pitchr1=[];
yawr1=[];
rollc2=[];
pitchc2=[];
yawc2=[];
rolll2=[];
pitchl2=[];
yawl2=[];
rollr2=[];
pitchr2=[];
yawr2=[];
ac1=[];
ac2=[];

% Create a loop that takes the time (currently a string) and converts
% it into a number that represents the time in seconds. This for loop
% also takes the data and computes the quaternion and Euler angles
% using the FQA and Euler functions developed by Dr. Calusdian and
% Dr. Yun.

% This loop is for the chest sensor

for aa=1:length(chest.textdata(:,2))
    firstc=chest.textdata(aa,1);
    thirdc=chest.textdata(aa,3);
    forthc=chest.textdata(aa,4);
    fifthc=chest.textdata(aa,5);
    hourc=firstc{1}(10:11);
    minutec=firstc{1}(13:14);
    secondc=firstc{1}(16:24);
    hourc1(aa)=str2num(hourc);
    minutec1(aa)=str2num(minutec);
    secondc1(aa)=str2num(secondc);
    timec(aa)=hourc1(aa)*3600+minutec1(aa)*60+secondc1(aa);
    accelcx=thirdc{1}((end-7):end);
    accelcy=forthc{1}(1:end);
    accelcz=fifthc{1}(1:8);
    compxc=fifthc{1}((end-7):end);
    accelxc1(aa)=str2num(accelcx); % x-acceleration
    accelyc1(aa)=str2num(accelcy); % y-acceleration
    accelzc1(aa)=str2num(accelcz); % z-acceleration
    compxc1(aa)=str2num(compxc);   % x-Magnetometer reading
end
compyc1=chest.data(:,1)'; % y-Magnetometer reading
```

```
compzc1=chest.data(:,2)'; % z-Magnetometer reading
% Puts all accelerations and Magnetometer readings into vector
ac=[accelxc1;accelyc1;accelzc1];
mc=[compxc1;compyc1;compzc1];

% This for loop takes the acceleration and magnetometer readings and
% converts them to a quaternion. The quaternion is then used to
% extract the Euler angles.

for bb=1:length(chest.textdata(:,2))
    [qc]=fqa(ac(:,bb),mc(:,bb));
    EAc=[EAc, Euler(qc)];
end

% Repeat this for the other two sensors
% This loop is for the left leg

for cc=1:length(left.textdata(:,2))
    firstlf=left.textdata(cc,1);
    thirdlf=left.textdata(cc,3);
    forthlf=left.textdata(cc,4);
    fifthlf=left.textdata(cc,5);
    hourlf=firstlf{1}(10:11);
    minutelf=firstlf{1}(13:14);
    secondlf=firstlf{1}(16:24);
    hourlf1(cc)=str2num(hourlf);
    minutelf1(cc)=str2num(minutelf);
    secondlf1(cc)=str2num(secondlf);
    timelf(cc)=hourlf1(cc)*3600+minutelf1(cc)*60+secondlf1(cc);
    accellfx=thirdlf{1}((end-7):end);
    accellfy=forthlf{1}(1:end);
    accellfz=fifthlf{1}(1:8);
    compxlf=fifthlf{1}((end-7):end);
    accelxlf1(cc)=str2num(accellfx); % x-acceleration
    accelylf1(cc)=str2num(accellfy); % y-acceleration
    accelzlf1(cc)=str2num(accellfz); % z-acceleration
    compxlf1(cc)=str2num(compxlf);   % x-Magnetometer reading
end
compylf1=left.data(:,1)'; % y-Magnetometer reading
compzlf1=left.data(:,2)'; % z-Magnetometer reading
% Puts all accelerations and Magnetometer readings into vector
al=[accelxlf1;accelylf1;accelzlf1];
ml=[compxlf1;compylf1;compzlf1];
% This for loop takes the acceleration and magnetometer readings and
% converts them to a quaternion. The quaternion is then used to extract
% the Euler angles.
for dd=1:length(left.textdata(:,2))
    [ql]=fqa(al(:,dd),ml(:,dd));
    EAl=[EAl, Euler(ql)];
end

% This loop is for the right leg

for ee=1:length(right.textdata(:,2))
```

```
    firstrf=right.textdata(ee,1);
    thirdrf=right.textdata(ee,3);
    forthrf=right.textdata(ee,4);
    fifthrf=right.textdata(ee,5);
    hourrf=firstrf{1}(10:11);
    minuterf=firstrf{1}(13:14);
    secondrf=firstrf{1}(16:24);
    hourrf1(ee)=str2num(hourrf);
    minuterf1(ee)=str2num(minuterf);
    secondrf1(ee)=str2num(secondrf);
    timerf(ee)=hourrf1(ee)*3600+minuterf1(ee)*60+secondrf1(ee);
    accelrfx=thirdrf{1}((end-7):end);
    accelrfy=forthrf{1}(1:end);
    accelrfz=fifthrf{1}(1:8);
    compxrf=fifthrf{1}((end-7):end);
    accelxrf1(ee)=str2num(accelrfx); % x-acceleration
    accelyrf1(ee)=str2num(accelrfy); % y-acceleration
    accelzrf1(ee)=str2num(accelrfz); % z-acceleration
    compxrf1(ee)=str2num(compxrf);   % x-Magnetometer reading
end
compyrf1=right.data(:,1)'; % y-Magnetometer reading
compzrf1=right.data(:,2)'; % z-Magnetometer reading
ar=[accelxrf1;accelyrf1;accelzrf1];
mr=[compxrf1;compyrf1;compzrf1];

% This for loop takes the acceleration and magnetometer readings and
% converts them to a quaternion. The quaternion is then used to extract
% the Euler angles.

for ff=1:length(right.textdata(:,2))
    [qr]=fqa(ar(:,ff),mr(:,ff));
    EAr=[EAr, Euler(qr)];
end

%  Re-name the angles and convert to degrees

rollc=EAc(1,:)*180/pi;
pitchc=EAc(2,:)*180/pi;
yawc=EAc(3,:)*180/pi;

rolll=EAl(1,:)*180/pi;
pitchl=EAl(2,:)*180/pi;
yawl=EAl(3,:)*180/pi;

rollr=EAr(1,:)*180/pi;
pitchr=EAr(2,:)*180/pi;
yawr=EAr(3,:)*180/pi;

% Find the last sensor turned on and use this as a base time of zero
% seconds. This is necessary because all of the sensors are not turned
% on at the same time. This will ensure that all of the sensors are
% synced.

timeref=max([timec(1),timelf(1),timerf(1)]);
```

```
timec=timec-timeref;
timelf=timelf-timeref;
timerf=timerf-timeref;

% Define alpha for the gain of the low-pass filter to be implemented in
% the next loops. The low-pass filter is as follows: for input X
% and output Y, Y(k)=alpha*X(k)+(1-alpha)*Y(k-1). The purpose of this
% filter is to smooth out some of the rapid changes in acceleration and
% angles.

alpha=.85;

% These for loops truncate any data taken before the reference time
% (time last sensor turned on). It also implements the low-pass filter
% to smooth the data. The first loop is for the chest sensor.

for gg=1:length(timec)
    % only take data for times greater than zero
    if timec(gg)>=0
        timec1=[timec1,timec(gg)];
        rollc1=[rollc1, rollc(gg)];
        pitchc1=[pitchc1, pitchc(gg)];
        yawc1=[yawc1, yawc(gg)];
        ac1=[ac1, ac(3,gg)];
    end
end

% Implement low-pass filter

for gg=1:length(timec1)
        if gg==1
            rollc2=[rollc2, rollc1(gg)];
            pitchc2=[pitchc2, pitchc1(gg)];
            yawc2=[yawc2, yawc1(gg)];
            ac2=[ac2, ac1(gg)];
        else
            rollc2=[rollc2, (1-alpha)*rollc1(gg)+alpha*rollc2(gg-1)];
            pitchc2=[pitchc2, (1-alpha)*pitchc1(gg)+alpha*pitchc2(gg-
1)];
            yawc2=[yawc2, (1-alpha)*yawc1(gg)+alpha*yawc2(gg-1)];
            ac2=[ac2, (1-alpha)*ac1(gg)+alpha*ac2(gg-1)];
        end
end

% Another loop for the left leg sensor

for hh=1:length(timelf)
    % only take data for times greater than zero
    if timelf(hh)>=0
        timelf1=[timelf1, timelf(hh)];
        rolll1=[rolll1, rolll(hh)];
        pitchl1=[pitchl1, pitchl(hh)];
        yawl1=[yawl1, yawl(hh)];
    end
```

```
end

% Implement low-pass filter

for hh=1:length(timelf1)
    if hh==1
        rolll2=[rolll2, rolll1(hh)];
        pitchl2=[pitchl2, pitchl1(hh)];
        yawl2=[yawl2, yawl1(hh)];
    else
        rolll2=[rolll2, (1-alpha)*rolll1(hh)+alpha*rolll2(hh-1)];
        pitchl2=[pitchl2, (1-alpha)*pitchl1(hh)+alpha*pitchl2(hh-1)];
        yawl2=[yawl2, (1-alpha)*yawl1(hh)+alpha*yawl2(hh-1)];
    end
end

% Another loop for the right leg

for ii=1:length(timerf)
    % only take data for times greater than zero
    if timerf(ii)>=0
        timerf1=[timerf1, timerf(ii)];
        rollr1=[rollr1, rollr(ii)];
        pitchr1=[pitchr1, pitchr(ii)];
        yawr1=[yawr1, yawr(ii)];
    end
end

% Implement low-pass filter

for ii=1:length(timerf1)
    if ii==1
        rollr2=[rollr2, rollr1(ii)];
        pitchr2=[pitchr2, pitchr1(ii)];
        yawr2=[yawr2, yawr1(ii)];
    else
        rollr2=[rollr2, (1-alpha)*rollr1(ii)+alpha*rollr2(ii-1)];
        pitchr2=[pitchr2, (1-alpha)*pitchr1(ii)+alpha*pitchr2(ii-1)];
        yawr2=[yawr2, (1-alpha)*yawr1(ii)+alpha*yawr2(ii-1)];
    end
end

% These variables define the parameters to be changed in order to
% adjust the number of false alarms and missed data. The first_point
% variable is defining the amount of points at the beginning of the
% data used to compute the average starting or reference angle.
% The stabilizetime variable is to set the number of data points that
% will be averaged in order to determine the angular change from the
% reference angle.
% The angle_threshold variable is the minimum angle that the sensor
% must deviate from the reference angle before it is considered
% significantly different or outside of the normal motion of walking
% and running.
```

```
first_point=40;
chest_ang_ref=mean(pitchc(1:first_point));
left_ang_ref=mean(pitchl(1:first_point));
right_ang_ref=mean(pitchr(1:first_point));
stabilizetime=30;
angle_threshold=50;

% The purpose of the minimum variable is to truncate the end of the
% data to the shortest sample due to the fact that they are turned off
% at different times.

minimum=min([length(timec1),length(timelf1),length(timerf1)]);

% For the following loops starting with the point after "first_point,"
% the previous 30 points are averaged and the reference angle is
% subtracted. The absolute value is then taken of the mean change from
% the reference angle. The pose is then computed as standing/walking,
% kneeling, or prone.

for jj=first_point:minimum
    if abs(mean(pitchl2(jj-stabilizetime:jj))-left_ang_ref)...
            >angle_threshold||abs(mean(pitchr2(jj-stabilizetime:jj)))...
            -right_ang_ref)>angle_threshold
        % if either leg is down then the pose is -1 or kneeling
        pose(jj)=-1;
        if abs(mean(pitchc2(jj-stabilizetime:jj))...
                -chest_ang_ref)>angle_threshold+10
            % if the chest has also fallen then the pose is -2 or prone
            pose(jj)=-2;
        end
    else
        % if neither leg is down then the pose is 0 or walking/standing
        pose(jj)=0;
    end

    % This is where the hysteresis control is implemented. This logic
    % says that if we are in kneeling or prone, then we have to have an
    % angular change back towards the reference angle surpassing the
    % angle that was determined to get into that pose. For example, if
    % the chest sensor needs to read reference angle - 60 degrees to
    % register prone, then it needs to read reference angle - 40
    % degrees to get out of prone

    if pose(jj-1)==-1
        if abs(mean(pitchl2(jj-stabilizetime:jj))-left_ang_ref)...
                >(angle_threshold-25)||...
                abs(mean(pitchr2(jj-stabilizetime:jj))-
right_ang_ref)...
                >(angle_threshold-25)
            pose(jj)=-1;
            if abs(mean(pitchc2(jj-stabilizetime:jj))...
                    -chest_ang_ref)>angle_threshold+10
                pose(jj)=-2;
            end
```

```
        end
    elseif pose(jj-1)==-2
        if abs(mean(pitchc2(jj-stabilizetime:jj))...
                -chest_ang_ref)>angle_threshold-20
            pose(jj)=-2;
        end
    end
end

% The following two loops identify times while standing where we are
% running. If standing or just walking, the algorithm currently allows
% data collection for PNS.

% For any points where the downward acceleration in the chest sensor is
% greater than 1.4 G's, and the current pose is standing/walking, then
% a peak variable is created. The value of 1.4 G's was found
% experimentally.

for ll=1:length(pose)
    if ll<40
        pose(ll)=pose(ll);
        peak(ll)=0;
    else
        if pose(ll)==0&&ac2(ll)<=-1.4
            peak(ll)=1;
        else
            peak(ll)=0;
        end
    end
end

% If there is more than 1 peak that occurs within 30 time steps then it
% can be considered running. 30 time steps was based off the frequency
% of footsteps while running vice walking or another activity.

for mm=1:length(peak)
    if mm<31
        run(mm)=0;
    elseif sum(peak(mm-30:mm))>1
        run(mm:mm+3)=1;
    else
        run(mm)=0;
    end
    if run(mm)==1
        pose(mm)=1;
    end
end

% I now calculate the time for use in plotting the "pose" variable

time_pose=timec1(1:minimum);

save([nameOfFile,'_pose.mat'],'pose','time_pose','timeref')
```

```
figure(1)
subplot(3,1,1)
plot(timec1(1:minimum),pitchc2(1:minimum),'g')
grid on
hold on
plot(time_pose(1:minimum),pose(1:minimum)*100,'*k','Markersize',.5)
hold off
title('Chest Pitch and Pose')
ylim([-220,120])
ax=gca;
ax.YTick=[-200 -100 0 100];
ax.YTickLabel={'Prone','Kneeling','Walking/Standing','Running'};
xlabel('Time(s)')
subplot(3,1,2)
plot(timelf1(1:minimum),pitchl2(1:minimum),'g')
grid on
hold on
plot(time_pose(1:minimum),pose(1:minimum)*100,'*k','Markersize',.5)
hold off
title('Left Leg Pitch and Pose')
ylim([-220,120])
ax=gca;
ax.YTick=[-200 -100 0 100];
ax.YTickLabel={'Prone','Kneeling','Walking/Standing','Running'};
xlabel('Time(s)')
subplot(3,1,3)
plot(timerf1(1:minimum),pitchr2(1:minimum),'g')
grid on
hold on
plot(time_pose(1:minimum),pose(1:minimum)*100,'*k','Markersize',.5)
hold off
title('Right Leg Pitch and Pose')
ylim([-220,120])
ax=gca;
ax.YTick=[-200 -100 0 100];
ax.YTickLabel={'Prone','Kneeling','Walking/Standing','Running'};
xlabel('Time(s)')
```

2. MATLAB Implementation of the FQA Function

```
function [q, error, flag] = fqa(a, m);
% a is 3x1, m is 3x1

% Xiaoping Yun, May 7, 2008
% input a = 3-dim acceleration, m=3-dim local magnetic measurement

%Mref = [0.4943  0.0  0.8693];

epsilon = 0.10; % accuracy control constant
singular_flag = 0;
alpha = 30*pi/180; % offset angle
```

```
% x is 3x2, first col = magnetometer, second col = accelerometer.

a_bar = a/norm(a); % make sure that it is normalized
m_b = m/norm(m);

sin_th = a_bar(1);
cos_th = sqrt(1-sin_th^2);

% singularity avoidance algorithm
if (cos_th <= epsilon)
    singular_flag = 1;
    q_offset =cos(alpha/2)*[1 0 0 0]' + sin(alpha/2)*[0 0 1 0]';

    a_bar_q = [0; a_bar];
    m_b_q = [0; m_b];
    a_q_offset = rotate_v_by_q(a_bar_q,q_offset);
    m_q_offset = rotate_v_by_q(m_b_q,q_offset);

    a_bar = a_q_offset(2:4);
    m_b = m_q_offset(2:4);

else
   % do not do anything other than setting the flag.
   singular_flag = 0;
end



% elevation quaternion
sin_th = a_bar(1);%h(1);
% cos_th = sqrt(1-sin_th^2);
cos_th = sqrt(a_bar(2)^2 + a_bar(3)^2); %J.C. 1/30/2009

% computing half-angle values
cos_half_th=sqrt((1+cos_th)/2);
if (cos_th<=-1) % this "if" is needed since sign(0) = 0.
    sin_half_th = 1;
else
    sin_half_th=sign(sin_th)*sqrt((1-cos_th)/2);
end


qe = cos_half_th*[1;0;0;0] + sin_half_th*[0;0;1;0];

% Roll Quaternion
b = [a_bar(2) a_bar(3)];
c = b/norm(b);
sin_phi = -c(1);
cos_phi = -c(2);


cos_half_phi=sqrt((1+cos_phi)/2);
if (cos_phi<=-1)
    sin_half_phi = 1;
```

```
else
    sin_half_phi=sign(sin_phi)*sqrt((1-cos_phi)/2);
end


qr = cos_half_phi*[1;0;0;0] + sin_half_phi*[0;1;0;0];

% Azimuth Quaternion

qe_inv = [qe(1);-qe(2);-qe(3);-qe(4)];
qr_inv = [qr(1);-qr(2);-qr(3);-qr(4)];
m_b_q = [0; m_b];

q_er = q_mult2(qe,qr);
q_er_inv =[q_er(1); -q_er(2); -q_er(3); -q_er(4)];
m_e = q_mult2(q_er,q_mult2(m_b_q, q_er_inv));

M = [m_e(2),m_e(3)];
M = M/norm(M);
N = [1; 0];
tmp =  [ M(1) M(2);
        -M(2) M(1)]*N;
cos_psi = tmp(1);
sin_psi = tmp(2);

cos_half_psi=sqrt((1+cos_psi)/2);
if (cos_psi<=-1) % IMPORTANT, if it is written as cos_psi==-1, it does
% not work. cos_psi is potentially less than -1.
    sin_half_psi = 1;
else
     sin_half_psi=sign(sin_psi)*sqrt((1-cos_psi)/2);
end

qa = cos_half_psi*[1;0;0;0]+ sin_half_psi*[0;0;0;1];

q_tmp1 = q_mult2(qe,qr);
q_tmp = q_mult2(qa,q_tmp1);

if (singular_flag == 1)
    q = q_mult2(q_tmp, q_offset);
else
    q = q_tmp;
end

error = cos_th;
flag = singular_flag;
```

3. MATLAB Implementation of the Euler Function

```
function EulerAngles=Euler(u)
```

```
q0=u(1);
q1=u(2);
q2=u(3);
q3=u(4);

B=[q0^2+q1^2-q2^2-q3^2 2*(q1*q2+q3*q0) 2*(q1*q3-q0*q2);
   2*(q1*q2-q0*q3) q0^2-q1^2+q2^2-q3^2 2*(q2*q3+q0*q1);
   2*(q1*q3+q0*q2) 2*(q2*q3-q0*q1) q0^2-q1^2-q2^2+q3^2];

phi=atan2(B(2,3),B(3,3));
theta=-asin(B(1,3));
psi=atan2(B(1,2),B(1,1));

EulerAngles=[phi;
             theta;
             psi];
```

4. MATLAB Quaternion Multiply Function

This function is called by the FQA function.

```
function qout=q_mult2(p,q)

P_mat = [p(1) -p(2) -p(3) -p(4);
         p(2)  p(1) -p(4)  p(3);
         p(3)  p(4)  p(1) -p(2);
         p(4) -p(3)  p(2)  p(1)];
qout = P_mat*q;
```

5. MATLAB Function for Rotation by a Quaternion

This function is called by the FQA function.

```
function u=rotate_v_by_q(v,q)

q_inv= [q(1) -q(2) -q(3) -q(4)]';

u = q_mult2(q,q_mult2(v,q_inv));
```

B. MATLAB CODE FOR REAL-TIME IMPLEMENTATION OF THE POSTURE-TRACKING SYSTEM

The following code is used for the real-time implementation of the posture-tracking system. The FQA, Euler, quaternion multiply, and quaternion rotation functions are all still required for use in the following code.

1. Real-Time Posture Tracking Main Algorithm

```
clear all
clc
% the following establishes commands to be sent to the YEI sensor to
% set up streaming.

TSS_START_BYTE = uint8(247);
TSS_RESPONSE_HEADER_START_BYTE = uint8(249);
TSS_START_STREAMING = uint8(85);
TSS_STOP_STREAMING = uint8(86);
TSS_TARE_CURRENT_ORIENTATION = uint8(96);
TSS_GET_UNTARED_EULERS = uint8(7);
TSS_NULL = uint8(255);
% s1, s2, and s3 sets the serial ports that the sensors are connected
% to. In this case, COM11 is the chest, COM13 is the left shin, and
% COM12 is the right shin.

s1 = serial('COM11','BaudRate', 115200,'ByteOrder','bigEndian');
fopen(s1); %opens the COM port

s2 = serial('COM13','BaudRate', 115200,'ByteOrder','bigEndian');
fopen(s2); %opens the COM port

s3 = serial('COM12','BaudRate', 115200,'ByteOrder','bigEndian');
fopen(s3); %opens the COM port

% sets stream to give data that is wanted. In this case they ask for
% accel and compass data. See next appendix B-2.
setupstreaming_header_s1
setupstreaming_header_s2
setupstreaming_header_s3

% With parameter-less wired commands the command byte will be the same
% as the checksum
write_bytes = [];
write_bytes = [write_bytes, TSS_RESPONSE_HEADER_START_BYTE,
TSS_START_STREAMING, TSS_START_STREAMING]; %tell the sensor to stream

% sends all information about header data as well as what information
% is requested to the sensor.
fwrite(s1, write_bytes,'uint8');
fwrite(s2, write_bytes,'uint8');
fwrite(s3, write_bytes,'uint8');
```

```
% all of the following section of commands pre-allocates the size of
each of the pieces of data for the posture-tracking algorithm to use.

sample_count = 1;
n= 10000;
z_accel_1=zeros(1,n);
pitch_1 = zeros(1,n);
pitch_2 = zeros(1,n);
pitch_3 = zeros(1,n);
z_accel_1_filtered=zeros(1,n);
timestamp1 = zeros(1,n);
timestamp2 = zeros(1,n);
timestamp3 = zeros(1,n);
pose=zeros(1,n);

while sample_count< n

      % the following if statements throws away the header information
      % for the first readings and gets the accel and compass data for
      % it. The algorithm for posture-tracking is then implemented.

    if sample_count == 1 %required to throw out extraneous header data
% received from instruction set
        disp('START')
        throw_out_header1 = fread(s1,2,'uint32');
        throw_out_data1 = fread(s1,6,'single');
        throw_out_header2 = fread(s2,2,'uint32');
        throw_out_data2 = fread(s2,6,'single');
        throw_out_header3 = fread(s3,2,'uint32');
        throw_out_data3 = fread(s3,6,'single');
    elseif sample_count==2
        timestamp_micro1 = fread(s1,1,'uint32'); %reads the time stamp,
        data_str1 = fread(s1,6,'single'); %3 axis accelerometer
                                          % readings and 3 magnetometer
        timestamp_micro2 = fread(s2,1,'uint32'); %reads the time stamp,
        data_str2 = fread(s2,6,'single'); %3 axis accelerometer
                                          % readings and 3 magnetometer
        timestamp_micro3 = fread(s3,1,'uint32'); %reads the time stamp,
        data_str3 = fread(s3,6,'single'); %3 axis accelerometer
                                          % readings and 3 magnetometer


        x_accel_1 = data_str1(1) ;
        y_accel_1 = data_str1(2) ;
        z_accel_1(sample_count-1) = data_str1(3);
        x_compass_1 = data_str1(4);
        y_compass_1 = data_str1(5);
        z_compass_1 = data_str1(6);
        timestamp1(sample_count-1) = timestamp_micro1;
        accel_1 = [x_accel_1;y_accel_1; z_accel_1(sample_count-1)];
        compass_1 = [x_compass_1;y_compass_1;z_compass_1];

        x_accel_2= data_str2(1) ;
        y_accel_2 = data_str2(2) ;
```

```
    z_accel_2 = data_str2(3);
    x_compass_2 = data_str2(4);
    y_compass_2 = data_str2(5);
    z_compass_2 = data_str2(6);
    timestamp2(sample_count-1) = timestamp_micro2;
    accel_2 = [x_accel_2;y_accel_2; z_accel_2];
    compass_2 = [x_compass_2;y_compass_2;z_compass_2];

    x_accel_3 = data_str3(1) ;
    y_accel_3= data_str3(2) ;
    z_accel_3= data_str3(3);
    x_compass_3 = data_str3(4);
    y_compass_3 = data_str3(5);
    z_compass_3= data_str3(6);
    timestamp3(sample_count-1) = timestamp_micro3;
    accel_3 = [x_accel_3;y_accel_3; z_accel_3];
    compass_3 = [x_compass_3;y_compass_3;z_compass_3];

    [q1]=fqa(accel_1,compass_1);
    [E1]=Euler(q1);
    pitch_1(sample_count-1) = E1(2)*180/pi;
    z_accel_1_filtered(sample_count-1)=z_accel_1(sample_count-1);

    [q2]=fqa(accel_2,compass_2);
    [E2]=Euler(q2);
    pitch_2(sample_count-1) = E2(2)*180/pi;

    [q3]=fqa(accel_3,compass_3);
    [E3]=Euler(q3);
    pitch_3(sample_count-1) = E3(2)*180/pi;


% this elseif statement gets the time, accel, and compass data for
% each timestep after the first one. It then implements the
% posture-tracking algorithm.
elseif sample_count>2
%now that extraneous header data on first return from sensor is
% gone, actually obtain accelerometer readings and time
% stamp(needed for analytic purposes)
    timestamp_micro1 = fread(s1,1,'uint32'); %reads the time stamp,
    data_str1 = fread(s1,6,'single'); %3 axis accelerometer
                                      % readings and 3 magnetometer
    timestamp_micro2 = fread(s2,1,'uint32'); %reads the time stamp,
    data_str2 = fread(s2,6,'single'); %3 axis accelerometer
                                      % readings and 3 magnetometer
    timestamp_micro3 = fread(s3,1,'uint32'); %reads the time stamp,
    data_str3 = fread(s3,6,'single'); %3 axis accelerometer
                                      % readings and 3 magnetometer

    x_accel_1 = data_str1(1) ;
    y_accel_1 = data_str1(2) ;
    z_accel_1(sample_count-1) = data_str1(3);
    x_compass_1 = data_str1(4);
    y_compass_1 = data_str1(5);
```

```
        z_compass_1 = data_str1(6);
        timestamp1(sample_count-1) = timestamp_micro1;
        accel_1 = [x_accel_1;y_accel_1; z_accel_1(sample_count-1)];
        compass_1 = [x_compass_1;y_compass_1;z_compass_1];

        x_accel_2= data_str2(1) ;
        y_accel_2 = data_str2(2) ;
        z_accel_2 = data_str2(3);
        x_compass_2 = data_str2(4);
        y_compass_2 = data_str2(5);
        z_compass_2 = data_str2(6);
        timestamp2(sample_count-1) = timestamp_micro2;
        accel_2 = [x_accel_2;y_accel_2; z_accel_2];
        compass_2 = [x_compass_2;y_compass_2;z_compass_2];

        x_accel_3 = data_str3(1) ;
        y_accel_3= data_str3(2) ;
        z_accel_3= data_str3(3);
        x_compass_3 = data_str3(4);
        y_compass_3 = data_str3(5);
        z_compass_3= data_str3(6);
        timestamp3(sample_count-1) = timestamp_micro3;
        accel_3 = [x_accel_3;y_accel_3; z_accel_3];
        compass_3 = [x_compass_3;y_compass_3;z_compass_3];

        [q1]=fqa(accel_1,compass_1);
        [E1]=Euler(q1);
        pitch_1(sample_count-1) = (.85*pitch_1(sample_count-
2)+.15*E1(2)*180/pi);
        z_accel_1_filtered(sample_count-
1)=(.85*z_accel_1_filtered(sample_count-2)+.15*z_accel_1(sample_count-
1));

        [q2]=fqa(accel_2,compass_2);
        [E2]=Euler(q2);
        pitch_2(sample_count-1) = (.85*pitch_2(sample_count-
2)+.15*E2(2)*180/pi);

        [q3]=fqa(accel_3,compass_3);
        [E3]=Euler(q3);
        pitch_3(sample_count-1) = (.85*pitch_3(sample_count-
2)+.15*E3(2)*180/pi);
    end
    if sample_count==40
        chestref=mean(pitch_1(1:sample_count));
        leftref=mean(pitch_2(1:sample_count));
        rightref=mean(pitch_3(1:sample_count));
    elseif sample_count>41
        pose(sample_count-1)=posture(pitch_1((sample_count-
41):(sample_count-1)),...
            pitch_2((sample_count-41):(sample_count-1)),...
            pitch_3((sample_count-41):(sample_count-1)),...
            z_accel_1_filtered((sample_count-41):(sample_count-1)),...
            chestref,leftref,rightref,...
```

```
            pose(sample_count-2));
        clc
        disp('Standing/Walking')
        if pose(sample_count-1)==0&&pose(sample_count-2)~=0
            clc
            disp('Standing/Walking')
        elseif pose(sample_count-1)==1&&pose(sample_count-2)~=1
            clc
            disp('Running')
        elseif pose(sample_count-1)==-1&&pose(sample_count-2)~=-1
            clc
            disp('Kneeling')
        elseif pose(sample_count-1)==-2&&pose(sample_count-2)~=-2
            clc
            disp('Prone')
        end
    end

    sample_count = sample_count+1;
end

% You must copy the following commands and paste them into the main
% MATLAB window before subsequently running the program. Failure to do
% this will cause the comport to lock, and MATLAB will require a
% restart before the program will run again.
fclose(s1)
fclose(s2)
fclose(s3)
```

2. MATLAB Code to Setup Streaming Data Collection for the Communications Port

The following MATLAB script file is used in the real-time posture-tracking code to setup header information for streaming and communicates with the comm ports. Three separate iterations of this script are required, one for each sensor. This script is titled setupstreaming_header_s1, setupstreaming_header_s2, and setupstreaming_header_s3.

```
%This program sets up a YEI 3 Space Sensor to stream raw acceleration
% data with a microsecond time stamp header

% Start Bytes, indicates the start of a command packet
TSS_START_BYTE = uint8(247);
TSS_SET_WIRED_RESPONSE_HEADER_BITFIELD = uint8(221);
GAP_BYTE = uint8(0);

% For a full list of streamable commands refer to "Wired Streaming
% Mode" section in the 3-Space Manual of your sensor
TSS_GET_CORRECTED_ACCEL_DATA = uint8(39);
```

```
TSS_GET_CORRECTED_MAGNETOMETER_DATA = uint8(40);
TSS_NULL = uint8(255);
% No command use to fill the empty slots in "set stream slots"

% For a full list of commands refer to the 3-Space Manual of your
% sensor
TSS_SET_STREAMING_SLOTS = uint8(80);
TSS_SET_STREAMING_TIMING = uint8(82);

%response header options-only will need microsecond time stamp
TSS_RH_SUCCESS_FAILURE = uint8(1);
TSS_RH_TIMESTAMP = uint8(2);
TSS_RH_COMMAND_ECHO = uint8(4);
TSS_RH_CHECKSUM = uint8(8);
TSS_RH_LOGICAL_ID = uint8(16);
TSS_RH_SERIAL_NUMBER = uint8(32);
TSS_RH_DATA_LENGTH = uint8(64);

disp('TSS_SET_UP_RESPONSE_HEADER')

write_bytes = [];
write_bytes = [write_bytes, TSS_START_BYTE];
write_bytes = [write_bytes, TSS_SET_WIRED_RESPONSE_HEADER_BITFIELD];
write_bytes = [write_bytes, GAP_BYTE, GAP_BYTE, GAP_BYTE,
TSS_RH_TIMESTAMP];

write_bytes_check_sum = write_bytes(2:length(write_bytes));
check_sum = uint8(mod((sum(write_bytes_check_sum)), 256));
%puts checksum into bits

write_bytes = [write_bytes, check_sum]; %add checksum byte

fwrite(s1, write_bytes,'uint8'); % sends the commands to the sensor.
% writes this opening set of bits to the designated serial port. Serial
% port must be designated and opened in main program


disp('TSS_SET_STREAMING SLOTS')

% The following commands build a packet of data that must be sent to
% the YEI sensor for data streaming. These commands are what decides
% where the data is streamed into.
% There are 8 streaming slots available for use, and each one can hold
% one of the streamable commands. Unused slots should be filled with
% TSS_NULL so that they will output nothing
write_bytes = [];
write_bytes = [write_bytes, TSS_START_BYTE];
write_bytes = [write_bytes, TSS_SET_STREAMING_SLOTS];
write_bytes = [write_bytes, TSS_GET_CORRECTED_ACCEL_DATA];
% stream slot0
write_bytes = [write_bytes, TSS_GET_CORRECTED_MAGNETOMETER_DATA];
% stream slot1
write_bytes = [write_bytes, TSS_NULL]; % stream slot2
```

```
write_bytes = [write_bytes, TSS_NULL]; % stream slot3
write_bytes = [write_bytes, TSS_NULL]; % stream slot4
write_bytes = [write_bytes, TSS_NULL]; % stream slot5
write_bytes = [write_bytes, TSS_NULL]; % stream slot6
write_bytes = [write_bytes, TSS_NULL]; % stream slot7

%calculating the checksum (note the checksum doesn't include the start
% byte) write_bytes.append((sum(write_bytes) - write_bytes[0]) % 256)
write_bytes_check_sum = write_bytes(2:length(write_bytes));
check_sum = uint8(mod((sum(write_bytes_check_sum)), 256));
%puts checksum into bits

write_bytes = [write_bytes, check_sum]; %at checksum byte

fwrite(s1, write_bytes,'uint8');
%writes this opening set of bits to the designated serial port. Serial
% port must be designated and opened in main program

disp('TSS_SET_STREAMING_TIMING')

% the following commands setup the packets to be sent to the sensor so
% that it can receive the time data.

%Interval determines how often the streaming session will output data
% from the requested commands An interval of 0 will output data at the
% max filter rate
 interval = uint8(0);% microseconds, needs to be 4 bytes long
%interval_byte3 = uint8(17);
%interval_byte4 = uint8(48);
%Duration determines how long the streaming session will run for
%A duration of 0xffffffff will have the streaming session run till the
% stop stream command is called
duration = uint8(255);%0xffffffff % microseconds 4 bytes long
% Delay determines how long the sensor should wait after a start
% command is issued to actually begin streaming
delay = uint8(0); % microseconds, 4 bytes long

write_bytes = [];
write_bytes = [write_bytes, TSS_START_BYTE];
write_bytes = [write_bytes, TSS_SET_STREAMING_TIMING];
write_bytes = [write_bytes, interval, interval, interval, interval];
write_bytes = [write_bytes, duration, duration, duration, duration];
write_bytes = [write_bytes, delay, delay, delay,delay];

%put into bigEndian
write_bytes = swapbytes(write_bytes);

%add check sum
write_bytes_check_sum = write_bytes(2:length(write_bytes));
check_sum = uint8(mod((sum(write_bytes_check_sum)), 256));
%puts checksum into bits

write_bytes = [write_bytes, check_sum]; %at checksum byte
```

```
fwrite(s1, write_bytes,'uint8');
%writes this opening set of bits to the designated serial port. Serial
% port must be designated and opened in main program
```

LIST OF REFERENCES

[1] C. K. Khan, "Geometry of fire tracking algorithm for direct fire weapon system," M.S. thesis, Dept. of Elect. and Comput. Eng., Naval Postgraduate School, Monterey, CA, 2015.

[2] J. C. Driesslein, "Scalable Mobile Ad Hoc Network (MANET) to enhance situational awareness in distributed small unit operations," M.S. thesis, Dept. of Elect. and Comput. Eng., Naval Postgraduate School, Monterey, CA, 2015.

[3] J. Calusdian, "A personal navigation system based on inertial and magnetic field measurements," Ph.D. dissertation, Dept. of Elect. and Comput. Eng., Naval Postgraduate School, Monterey, CA, 2010.

[4] C. Johnson, "A personal inertial navigation system based on multiple distributed nine degrees-of-freedom inertial measurement units," unpublished.

[5] A. Valtazanos, D. K. Arvind, and S. Ramamoorthy, "Using wearable inertial sensors for posture and position tracking in unconstrained environments through learned translation manifolds," in *Proceedings of the 12th International Conference on Information Processing in Sensor Networks*, ACM, New York, NY, 2013, pp. 241–252.

[6] F. Wenk and U. Frese, "Pose and posture estimation using inertial sensor data," in *Formal Modeling and Verification of Cyber-Physical Systems,* Springer Fachmedien Wiesbaden, 2015, pp. 308–310.

[7] X. Chen, "Human motion analysis with wearable inertial sensors," Ph.D. dissertation, Dept. of Philosophy, The University of Tennessee, Knoxville, TN, 2013.

[8] F. Yildiz, "Implementation of a human avatar for the MARG project in networked virtual environments," M.S. thesis, Modeling, Virtual Environments and Simulation (MOVES), Naval Postgraduate School, Monterey, CA, 2004.

[9] I. Pantazis, "Tracking human walking using MARG sensors," M.S. Thesis, Dept. of Elect. and Comput. Eng., Naval Postgraduate School, Monterey, CA, 2005.

[10] *Rifle Marksmanship*, Marine Corps Reference Publication 3–1A, USMC, 2012, pp. 6.10-6.30.

[11] 3-Space Sensor User's Manual, 1st ed., YEI Technology, Portsmouth, OH, 2014, pp. 4–47. [Online]. Available: http://www.yeitechnology.com/sites/default/files/YEI_TSS_Users_Manual_3.0_r1_4Nov2014.pdf.

[12] J. B. Kuipers, *Quaternions and Rotation Sequences*, 1st ed. Princeton, NJ: Princeton Univ. Press, 1999, pp. 83–86.

[13] X. Yun, E. R. Bachmann, and R. B. McGhee, "A simplified quaternion based algorithm for orientation estimation from Earth gravity and magnetic field measurements," *IEEE Transactions on Instrumentation and Measurement*, vol. 57, pp. 638–650, 2008.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California